VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

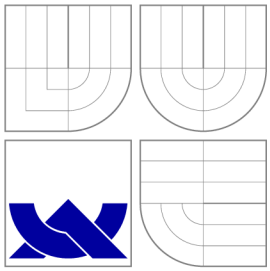
GENEROVÁNÍ BACKENDU APLIKACE Z UML MODELU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

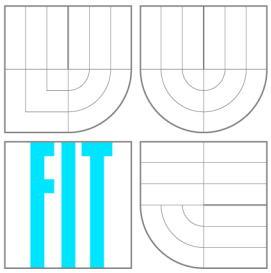
AUTOR PRÁCE
AUTHOR

VÁCLAV KLIKAR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENEROVÁNÍ BACKENDU APLIKACE Z UML MODELU

APPLICATION BACKEND GENERATION FROM A UML MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV KLIKAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2012

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat nástroj umožňující obousměrnou synchronizaci UML diagramu tříd a backendu aplikace kdykoliv v průběhu tvorby aplikace. Práce s UML modelem je umožněna pomocí standardizovaného XMI formátu. Backend aplikace je vytvořen a spravován prostřednictvím webového rámce Django. Pro čtení a zápis programového kódu Djanga je využit syntaktický analyzátor abstraktních syntaktických stromů.

Abstract

The aim of this bachelor's thesis is to design and implement a tool that allows bi-directional synchronization between a UML class diagram and an application backend whenever during application development. Working with the UML model is made possible by using a standardized XMI format. The application backend is created and managed through a web framework Django. To read and write Django program code, the tool uses abstract syntax tree parser.

Klíčová slova

Backend, administrace, webový rámec Django, UML diagram tříd, XMI, PySide, abstraktní syntaktický strom, ElementTree, generování kódu, synchronizace.

Keywords

Backend, administration, web framework Django, UML class diagram, XMI, PySide, abstract syntax tree, ElementTree, code generation, synchronization.

Citace

Václav Klikar: Generování backendu aplikace z UML modelu, bakalářská práce, Brno, FIT VUT v Brně, 2012

Generování backendu aplikace z UML modelu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Klikar
16. května 2012

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Radku Burgetovi, Ph.D. za možnost vypracovat tuto bakalářskou práci pod jeho odborným vedením a za cenné rady při její tvorbě. Dále bych chtěl poděkovat své rodině a přítelkyni za celkovou podporu při studiu.

Obsah

Kapitola 1

Úvod

V dnešní zrychlené době je čas velice vzácný, a proto i v oblasti softwarového inženýrství je snaha vytvářet produkty co nejrychleji a nejefektivněji. Postupně se přechází na programovací jazyky se stále větší mírou abstrakce či vznikají grafická uživatelská rozhraní umožňující aplikace tvořit i bez znalosti programování. Neméně významnou skupinou jsou pak webové aplikace, které se s rozvojem rychlosti a mobility připojení k internetu stávají den ode dne populárnějšími. Ty je vhodné vyvíjet tak, aby měly zvlášť oddělenou část zobrazovanou návštěvníkům webu (frontend) od části určené k administraci (backend).

Než ale bude moci být webová aplikace implementována v nějakém programovacím jazyce, je nejprve potřeba ji navrhnout. Jedním z prostředků pro návrh aplikací je grafický modelovací jazyk UML. Pomocí jej lze vytvářet různé typy digramů. Jedním z nich je i diagram tříd, který umožňuje vymodelovat strukturu systému dané aplikace.

Poté, co je návrh dokončený, vyvstává také otázka, jaké technologie zvolit pro samotnou implementaci. Kromě výběru programovacího jazyka je vhodné zvolit i nějaký webový aplikační rámec (framework), který bude sloužit jako nosná konstrukce vyvíjené aplikace. V současnosti jich je již k dispozici ohromné množství. Jedním takovým je i webový rámec Django, napsaný v jazyce Python. Ten mimo jiné umožňuje na základě definic datových modelů aplikace a dalších nastavení vytvořit s modely synchronizovanou databázi či vygenerovat komplexní administrační rozhraní.

Cílem této bakalářské práce je navrhnout a implementovat nástroj umožňující obousměrnou synchronizaci UML diagramu tříd a backendu aplikace kdykoliv v průběhu tvorby aplikace. Grafické uživatelské rozhraní vzniklého nástroje by mělo také umožnit backend aplikace různě konfigurovat. Vzhledem k tomu, že bývá zvykem UML diagram tříd v rámci návrhu aplikace tvořit, tak programátorovi aplikace díky tomuto nástroji odpadá nutnost manuálního přepisu UML modelu do programového kódu. Takto má programátor k dispozici automaticky vygenerovaný backend aplikace, který může z grafického rozhraní nadále parametrizovat.

Kapitola 2

Jazyk UML

Jazyk UML (Unified Modeling Language) je grafický modelovací jazyk, který je nejpoužívanějším jazykem v objektově orientovaných metodách návrhu systému. Umožňuje vytvářet několik typů diagramů, které lze rozdělit do skupin statického a dynamického modelu. Statický model zahrnuje diagramy modelující strukturu systému a dynamický model diagramy modelující chování systému. [13, 10]

2.1 Diagram tříd

Pro potřeby této bakalářské práce je podstatný pouze diagram tříd ze skupiny statického modelu. Diagram tříd slouží pro zobrazení struktury systému pomocí tříd a relací mezi nimi. Celá tato kapitola je založena na [13, 10, 9, 8], kde o této problematice můžete nalézt bližší informace. Primárně jsou v této práci uvedeny údaje o verzi jazyka UML 2.3 definované konsorciem OMG (Object Management Group) viz [9, 8]. Avšak specifikace ostatních verzí 2.x by měly také vyhovovat a je možné, např. v ukázkách zdrojového kódu této práce, spatřit definované i verze jiné nežli 2.3. V následujících podkapitolách 2.1.1 a 2.1.2 budou popsány základní prvky diagramu tříd pro účely této práce včetně potřebných specifikací z jádra metamodelu UML.

2.1.1 Třídy

Třída je definice množiny objektů, které sdílí stejné vlastnosti (více viz [13, 10]). Prvek jádra metamodelu UML definující třídu se nazývá `Class` a obsahuje tyto čtyři atributy:

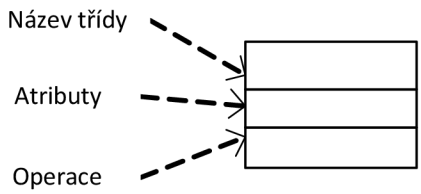
- `isAbstract: Boolean`: Při `true` je třída abstraktní. Výchozí je `false`.
- `ownedAttribute: Property[*]`: Atributy třídy (bez poděděných).
- `ownedOperation: Operation[*]`: Operace třídy (bez poděděných).
- `superClass: Class[*]`: Bezprostřední třídy, od kterých daná třída dědí.

Element `Property`, jež je typem pro `ownedAttribute`, reprezentuje atribut třídy. Jeho významnými atributy jsou `default: String[0..1]`, který určuje výchozí hodnotu atributu třídy, a `type: Type[0..1]` definující datový typ atributu třídy. Atribut `ownedOperation` je pro tuto práci nepodstatný.

Jednotlivé prvky v UML mohou být umísťovány do balíčků, které jsou pro ně jakýmsi kontejnerem s definovaným jmenným prostorem, kde všechny tyto prvky musí mít jedinečné

názvy. V jádře metamodelu UML se prvek definující balíček nazývá `Package` a ten sdružuje `/packagedElement: PackageableElement[*]`, které specifikují prvky v daném balíčku. Bližší informace ohledně specifikace třídy v jádru metamodelu UML naleznete v [8, 9].

V notaci UML má třída tvar obdélníku, jež je rozdělen na tři úseky umístěné pod sebou. Vrchní úsek je určen pro název třídy, prostřední úsek pro atributy třídy a spodní úsek pro operace třídy (viz obrázek 2.1).



Obrázek 2.1: Notace UML třídy

Formát atributu vypadá takto:

```
viditelnost název : typ = počáteční_hodnota
```

Název je jediný povinný údaj atributu třídy v jazyce UML. Viditelnost atributů určuje jejich přístupnost. Avšak u různých programovacích jazyků může mít viditelnost jiný význam, či mohou tyto jazyky definovat i viditelnosti, které nejsou v UML podporovány. Ve specifikaci jazyka UML jsou definovány čtyři základní typy viditelností:

- **„+" Veřejný (public)**: Viditelnost odkudkoli s přístupem k dané třídě.
- **„−" Soukromý (private)**: Viditelnost jen uvnitř dané třídy.
- **„#" Chráněný (protected)**: Viditelnost jen uvnitř dané třídy a jejích potomků.
- **„∼" Balíček (package)**: Viditelnost uvnitř téhož balíčku a jeho vnořených balíčků.

Typem atributu může být nějaká třída nebo primitivní typ. Ve specifikaci jazyka UML jsou definovány celkem čtyři primitivní typy – `Boolean`, `Integer`, `UnlimitedNatural` a `String`. Typ `Boolean` je reprezentován hodnotou `true` nebo `false`, `Integer` je číslo x, kde $x \in \mathbb{Z}$, `UnlimitedNatural` je číslo x, kde $x \in \mathbb{N}_0$ a `String` je řetězec znaků.

Operace a jiné možnosti tříd nejsou pro potřeby této bakalářské práce důležité. Pokud se o nich chcete něco dozvědět nebo získat více informací o problematice probrané výše, tak se můžete podívat do literatury viz [13, 10, 9].

2.1.2 Relace

Relace určují, jaký je vztah mezi třídami. Mezi relace, které jsou specifikovány v jazyce UML, patří asociace (association), agregace (aggregation), kompozice (composition), zobecnění (generalization), závislost (dependency) a realizace (realization). Notace těchto relací v jazyku UML a jejich stručný popis je uveden v tabulce 2.1.

Pro účely této práce je potřeba také znát některé informace ze specifikace jádra metamodelu UML pro asociaci a zobecnění. Prvek reprezentující asociaci se v jádře metamodelu UML nazývá `Association` a prvek popisující zobecnění se jmenuje `Generalization`. Prvek `Association` zahrnuje dvě podstatné části, a to `memberEnd: Property[2..*]` označující

koncové body asociace a `ownedEnd: Property[*]`, což jsou definice jednotlivých koncových bodů vlastněných danou asociací. Prvek `Generalization` obsahuje také dvě důležité části, kterými jsou `general: Classifier[1]` určující obecný klasifikátor (předka) a `specific: Classifier[1]` značící speciální klasifikátor (potomka). Více informací o specifikaci relací v jádru metamodelu UML lze najít v [8, 9].

Typ	**UML notace**	**Popis**
Asociace		Obecný vztah mezi třídami
Agregace		Specifičtější asociace, kdy celek je složen z částí
Kompozice		Silnější podoba agregace
Zobecnění		Dědičnost potomka od předka
Závislost		Změna v nezávislém objektu ovlivní závislý objekt
Realizace		Třída implementuje rozhraní

Tabulka 2.1: Přehled relací jazyka UML

Grafická notace relace může být v jazyce UML doplněna o některé další vlastnosti (viz [10]). Jednou z těchto vlastností je i násobnost relace. Ta udává počet objektů, které se mohou v libovolném okamžiku účastnit dané relace. Násobnost je zapisována jako interval ve formě `minimum..maximum` nebo konkrétní hodnotou. Např. násobnost `0..3` vyjadřuje „nula až tři", `1..*` znamená „jedna nebo více" či z `6` vyplývá „přesně šest". Hodnot může být i více, přičemž jsou oddělené čárkou, např. `3, 5, 7..9` znamená „tři nebo pět nebo sedm až devět". Pokud není násobnost uvedena, zůstává neurčitou. Pro bližší informace ohledně relací můžete nahlédnout do literatury viz [13, 10, 9].

2.2 Grafické nástroje pro tvorbu UML diagramů tříd

Grafických nástrojů pro tvorbu UML diagramů tříd existuje velké množství. U většiny z nich již nedochází k uvolňování nových verzí a jak už to bývá, tak ty nejkvalitnější jsou nástroje komerční, za které je nutno zaplatit. Při výběru UML nástroje pro potřeby této bakalářské práce nastaly dvě možnosti. Buď si zvolit jeden UML nástroj a pracovat s jeho vlastním vstupně/výstupním formátem souboru či zjistit, zda neexistuje nějaký standardizovaný formát souboru, se kterým by umělo pracovat větší množství UML nástrojů. Naštěstí takový formát souboru existuje a nazývá se XMI (viz kapitola 2.3).

Ovšem ani tak se vše neobejde bez problémů. Některé oblíbené UML nástroje s formátem XMI neumí dodnes vůbec pracovat (např. Dia 0.97.2) a jiné podporují pouze staré verze XMI 1.x (např. ArgoUML 0.34 stable, Umbrello UML Modeller 2.8.0 či StarUML 5.0.2). Avšak spousta nástrojů již umožňuje import a export novější verze XMI 2.1 z roku 2005 (viz [7]). Ta je podporována většinou komerčních UML nástrojů (např. Visual Paradigm for UML 9.0 nebo Enterprise Architect 9.3) i několika nástroji, které jsou dostupné zdarma (např. Modelio 2.1.1, ArgoUML 0.34 pre-alpha UML2 a Software Ideas Modeler 5.20). Některé nástroje také občas implementují XMI standard trochu po svém a není vždy možné

zaručit jeho přenositelnost.

2.3 XMI 2.1 standard

XMI (XML Metadata Interchange) je standard pro výměnu metadat definovaný konsorciem OMG (Object Management Group), který umožňuje vyjádřit objekty prostřednictvím jazyka XML (Extensible Markup Language), univerzálního formátu pro reprezentaci dat [11]. Pro získání základních informací ohledně XML se můžete podívat do literatury viz [11] nebo pro více detailů do nějakých odbornějších knih zabývajících se přímo jazykem XML. Metadata jsou standardizována také konsorciem OMG, a to pomocí MOF (Meta Object Facility), který umožňuje výměnu UML modelů mezi různými nástroji [11]. Verze XMI 2.1, ač není nejnovější, má v tuto chvíli mezi UML nástroji větší podporu nežli aktuální verze XMI 2.4.1.

V ukázce zdrojového kódu 2.1 lze vidět základní strukturu XMI dokumentu. Avšak některé nástroje pracují i s trochu odlišnou strukturou dokumentu, kdy neobsahují element `<xmi:XMI/>` a kořenovým elementem je přímo `<uml:Model/>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1"
    xmlns:uml="http://schema.omg.org/spec/UML/2.1"
    xmlns:xmi="http://schema.omg.org/spec/XMI/2.1">
    <uml:Model name="Example">
        ...
    </uml:Model>
</xmi:XMI>
```

Zdrojový kód 2.1: Základní struktura XMI dokumentu

Uvnitř elementu `<uml:Model/>` jsou definovány jednotlivé třídy a relace mezi nimi, případně další možné komponenty. Elementy, které je reprezentují mohou obsahovat některé XMI atributy. Významné z nich jsou [7]:

- `id`: Unikátní identifikátor elementu.
- `href`: Odkaz na `id` v daném XMI dokumentu nebo `id` v libovolném dokumentu, kde musí být za cestou k XMI dokumentu uvedené za znakem `#` požadované `id`.
  - např. takovýto `href="file2.xml#wanted"` nalezene element `<example xmi:id="wanted">` v souboru file2.xml
- `idref`: Odkaz na `id` v daném XMI dokumentu.
- `version`: Verze XMI specifikace.
- `type`: Specifikace typu elementu.
  - např. `xmi:type="uml:Class"` znamená, že se jedná o UML třídu, kde slovo `Class` je název prvku z jádra metamodelu UML specifikujícího třídu (viz kapitola 2.1.1).

Kapitola 3

Webový rámec Django

Django je webový rámec (framework) napsaný v jazyce Python. Zvládá objektově-relační mapování (ORM), kdy po definování datových modelů (viz níže kapitola 3.2) je možné používat automaticky vzniklé databázové API. Dále umožňuje generovat administrační rozhraní dle požadovaných vlastností (viz níže kapitola 3.3). Django má i spoustu jiných funkcí, avšak těmi se již tato práce nezabývá. Většina možností Djanga bude vyobrazena jen stručně, jelikož maximální rozsah této práce nedovoluje je detailněji popsat, i když pro účely této práce nebylo potřeba je všechny implementovat. Vytvořený nástroj implementuje jen ty možnosti Djanga, u nichž není nutné jejich funkčnost programovat pro specifické účely, ovlivňují backend aplikace a zároveň je jejich použití pro backend aplikace nepostradatelné či se jedná o de facto standardní vlastnosti, které by měly být umožněny parametrizovat. Celá tato kapitola je založena na [2, 12], kde můžete o Djangu nalézt bližší informace.

3.1 Správa rámce Django

Pro administrativní úkoly slouží v Djangu konzolové nástroje `django-admin.py` a `manage.py`, přičemž `manage.py` vše deleguje do `django-admin.py` a příkazy provádí jen v rámci již vytvořeného projektu. Nástroj `django-admin.py` je k nalezení v adresáři Djanga v cestě Python instalace a `manage.py` se nachází uvnitř každého Django projektu. Projekt je v Djangu vytvořen pomocí příkazu:

```
django-admin.py startproject <název_projektu>
```

Tímto je v dané složce, kde byl příkaz spuštěn, vygenerovaná adresářová struktura projektu pro zvolený název projektu, ve které se nachází soubory `__init__.py`, `manage.py`, `settings.py` a `urls.py`. Soubor `__init__.py` je prázdný soubor, dle kterého Python pozná, že adresář je balíček. V `settings.py` se nachází nastavení Django projektu a v `urls.py` jsou určeny jeho URL adresy.

Django umožňuje pracovat s různými databázovými systémy, přičemž výběr a přístupové údaje ke zvolené databázi se uvádí v souboru `settings.py`. Django má vestavěnou podporu pro databázové systémy PostgreSQL, SQLite 3, MySQL a Oracle, ale jsou pro něj dostupné i jiné databázové systémy. V `settings.py` je možné také zvolit časovou zónu, jazyk a případně další vlastnosti projektu Djanga.

Django obsahuje i vývojový server, který je již nakonfigurován pro daný projekt a lze spustit příkazem:

```
python manage.py runserver [port nebo adresa:port]
```

Při nezadání adresy či čísla portu je server spuštěn na IP adrese `127.0.0.1` a portu `8000`.

Projekt Django obsahuje nastavení celého webu a je složen z jedné či více webových aplikací. Aplikace se v projektu Django zakládá příkazem:

```
python manage.py startapp <název_aplikace>
```

V adresáři projektu Django je tímto vytvořena složka s názvem aplikace obsahující soubory `__init__.py`, `models.py`, `tests.py` a `views.py`. Soubor `models.py` popisuje datový model aplikace viz kapitola 3.2, `tests.py` je určen pro definování testů k aplikaci a `views.py` obsahuje pohledy, funkce zpracovávající požadavky prohlížeče a vracející mu odpovědi. Aplikace musí být také přidána do seznamu instalovaných aplikací v `settings.py`.

K synchronizaci projektu Django se zvolenou databází slouží příkaz:

```
python manage.py syncdb
```

Tento příkaz inicializuje zvolenou databázi pro daný projekt Django. Kontroluje seznam instalovaných aplikací a vytváří pro ně v databázi tabulky, pokud ještě neexistují. Pro modely jsou tak vytvořeny v databázi potřebné tabulky. Ovšem tímto příkazem je možné pouze tabulky inicializovat, nikoliv již měnit. Django nemá vestavěnou podporu pro migraci dat, takže v případě, kdy dojde k změně nějakého modelu je potřeba patřičně manuálně upravit i databázi. Tuto funkcionalitu však lze automaticky zajistit nějakou aplikací třetích stran umožňující migraci dat pro Django, jakou je například nástroj South viz kapitola 4.5.

3.2 Datové modely

Pomocí modelů jsou v Djangu popisována databázová data. Modely aplikace se nachází v souboru `models.py` uvnitř adresáře dané aplikace. Model je definován třídou Pythonu, která je potomkem `django.db.models.Model`. Každý model reprezentuje databázovou tabulku. Jednotlivá pole modelu jsou vytvářena jako atributy dané třídy a stanovují sloupce databázové tabulky. V ukázce zdrojového kódu 3.1 lze vidět, jakým způsobem se definuje model.

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=40)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()
```

Zdrojový kód 3.1: Příklad definice modelu

Typem pole modelu by měla být vhodná instance potomka třídy `django.db.models.Field`. Jmenovitě se jedná o typy `AutoField`, `BigIntegerField`, `BooleanField`, `CharField`, `CommaSeparatedIntegerField`, `DateField`, `DateTimeField`, `DecimalField`, `EmailField`, `FileField`, `FilePathField`, `FloatField`, `ImageField`, `IntegerField`, `IPAddressField`, `NullBooleanField`, `PositiveIntegerField`, `PositiveSmallIntegerField`, `SlugField`, `SmallIntegerField`, `TextField`, `TimeField` a `URLField`. Z většiny jejich názvů by mělo být patrné, jaká pole dané typy reprezentují. Pro jejich bližší specifikaci můžete nahlédnout do [2].

Jakákoliv pole mohou mít navíc nastavené volitelné volby definováním argumentů daných typů polí, jako je tomu v ukázce zdrojového kódu 3.1. Mezi volby, které jsou obsaženy i v této práci, patří `null`, `blank`, `choices`, `db_column`, `db_index`, `db_tablespace`,

`editable`, `help_text`, `primary_key` a `verbose_name`. Kromě těchto voleb platných pro všechny typy polí, lze u některých typů nastavit i typově specifické volby, které můžou být buď povinné nebo volitelné. Přehled těchto voleb je k vidění v tabulce 3.1.

Typ pole	Povinné argumenty	Volitelné argumenty
`CharField`	`max_length`	
`CommaSeparatedIntegerField`	`max_length`	
`DateField`		`auto_now` `auto_now_add`
`DateTimeField`		`auto_now` `auto_now_add`
`DecimalField`	`max_digits` `decimal_places`	
`EmailField`		`max_length`
`FileField`	`upload_to`	`max_length`

`FilePathField`	`path`	`match`
		`recursive`
		`max_length`

`ImageField`	`upload_to`	`height_field`
		`width_field`
		`max_length`

`SlugField`		`max_length`
`TimeField`		`auto_now` `auto_now_add`
`URLField`		`max_length`

Tabulka 3.1: Přehled typově specifických voleb pro typy polí

Kromě těchto typů polí existují v Djangu ještě relační pole, která reprezentují relace (viz kapitola 2.1.2). Jsou to `ForeignKey`, `ManyToManyField` a `OneToOneField`. Všechny tři vyžadují poziční argument, kterým je název modelu, s kterým jsou v relaci viz ukázka zdrojového kódu 3.2.

```
class Person(models.Model):
    ...

class BankAccount(models.Model):
    owner = models.ForeignKey("Person")
    ...
```

Zdrojový kód 3.2: Příklad definice relačního pole

3.3 Administrační rozhraní

Administrace je v Djangu automaticky generována dle nadefinovaných modelů pro administrační rozhraní, nacházejících se v souboru `admin.py` v adresáři aplikace. Pro její zpřístupnění z webového prohlížeče musí být v souboru `urls.py` administrace aktivována. Pro možnost přizpůsobení administrace požadovaným nastavením je potřeba v `admin.py` zaregistrovat nakonfigurované třídy, které jsou potomky

`django.contrib.admin.ModelAdmin` viz ukázka zdrojového kódu 3.3. Volby zahrnuté v této bakalářské práci, které umožňují měnit vlastnosti modelů v administračním rozhraní, jsou jmenovitě `actions_on_top`, `actions_on_bottom`, `actions_selection_counter`, `date_hierarchy`, `exclude`, `filter_horizontal`, `filter_vertical`, `list_display`, `list_editable`, `list_filter`, `list_per_page`, `radio_fields`, `raw_id_fields`, `save_as`, `save_on_top` a `search_fields`.

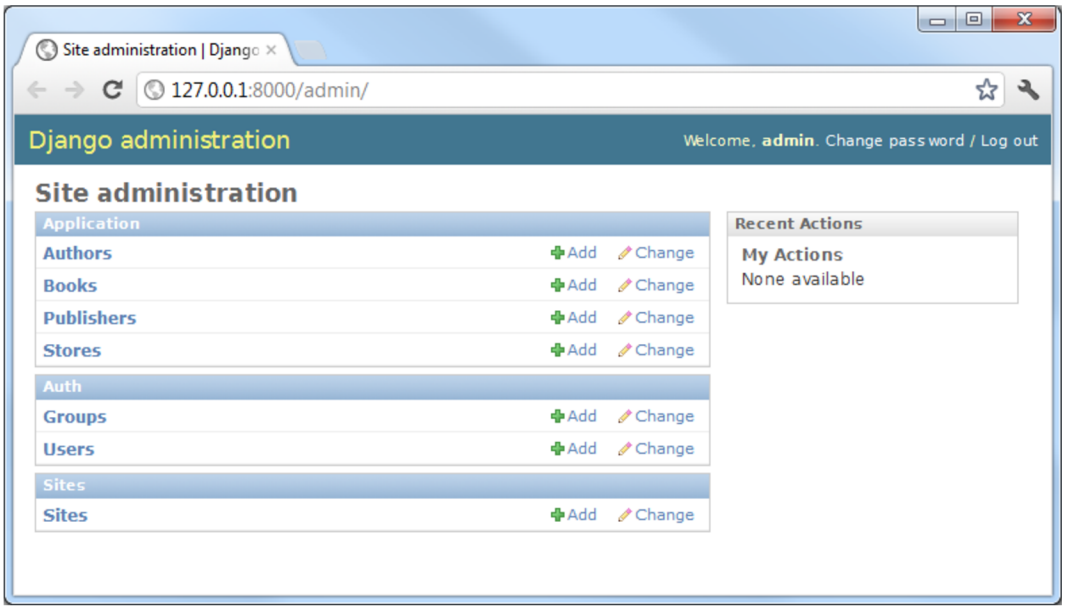
```
from django.contrib import admin
from models import Person

class PersonAdmin(admin.ModelAdmin):
    fields = ("first_name", "last_name")

admin.site.register(Person, PersonAdmin)
```

Zdrojový kód 3.3: Příklad nastavení modelu pro zobrazení v administraci

Vygenerované administrační rozhraní poskytuje přihlašovací formulář, kde po úspěšném přihlášení je zobrazena hlavní strana administrace viz obrázek 3.1. V administraci lze spravovat jednotlivé modely aplikací, uživatelské účty, uživatelské skupiny a weby. Jazyk administrace je zvolen dle nastavení jazyka projektu Django.



Obrázek 3.1: Ukázka automaticky vygenerovaného administračního rozhraní Django

Kapitola 4

Použité technologie

V této kapitole budou popsány technologie, které jsou v projektu použity. Také je zde zdůvodněno, proč byly právě tyto technologie vhodné pro tento projekt.

4.1 Programovací jazyk Python

Aplikace je napsána v jazyce Python verze 2.7. Jedním z hlavních důvodů volby tohoto jazyka a jeho verze bylo to, že je ve stejném jazyce a verzi napsán i webový rámec Django. Jazyk Python je v současné době rozšířen kromě verze 2.x i ve verzi 3.x, která ale není s 2.x zpětně kompatibilní. Proto by volba této novější verze nebyla ve výsledku moc praktická, když by při používání této aplikace byla na programátory používající rámec Django kladena nutnost mít nainstalovaný kromě interpretu jazyka Python 2.x i interpret jazyka Python 3.x. Dalším podstatným kladem při výběru jazyka Python byla také možnost mít aplikaci přenositelnou mezi různými operačními systémy vzhledem k tomu, že interpret Pythonu běží na systémech Windows, Linux/Unix i Mac OS X.

4.2 Rámec Qt prostřednictvím knihovny PySide

Jelikož jednou z hlavních částí této bakalářské práce je vznik grafického uživatelského rozhraní (z angl. Graphical User Interface, GUI), bylo také nutné zvolit vhodnou knihovnu umožňující v jazyce Python efektivně GUI programovat. Standardní GUI knihovnou jazyka Python je knihovna TkInter. Avšak k dispozici je velké množství i jiných knihoven. Pro GUI vytvořené aplikace je použit rámec Qt, a to prostřednictvím knihovny PySide. Kromě samotného rámce je k dispozici také nástroj Qt Designer, umožňující GUI navrhnout graficky. Umožňuje rozmístit komponenty GUI do formuláře včetně nastavení jejich vlastností a propojit signály a sloty dle potřeby. K vyslání signálu dojde při vzniku nějaké události na dané komponentě, a ten je poté zachycen přiřazeným slotem. Slotem je metoda, která zpracuje signál. Takto navržené GUI je poté pomocí nástroje pyside-uic převedeno do kódu jazyka Python. Kromě knihovny PySide existuje pro rámec Qt ještě jedna obdobně použitelná knihovna jménem PyQt. Avšak PySide má na rozdíl od PyQt volnější licenci pro její použití (LGPL verze 2.1) a záštitu nad ní má přímo firma Nokia, která je i současným vlastníkem rámce Qt.

4.3 ElementTree XML API pro syntaktickou analýzu XMI

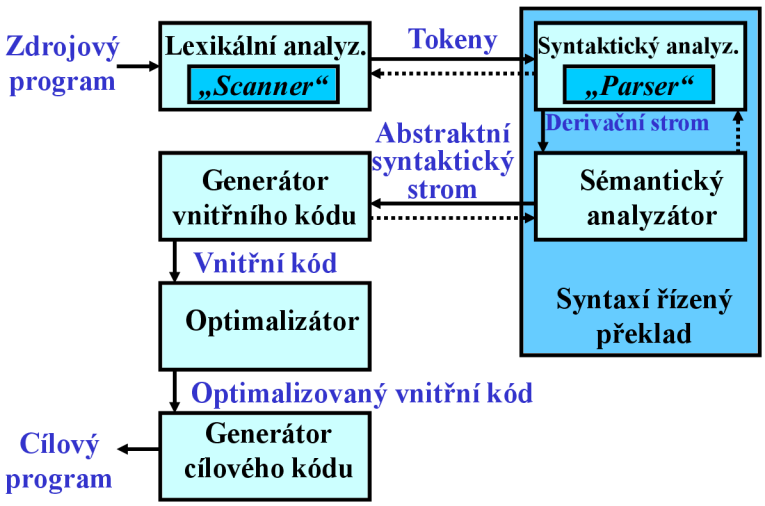
ElementTree XML API je součástí standardní knihovny jazyka Python od verze 2.5. Umožňuje analyzovat XML soubor a převést ho do instance třídy `ElementTree` reprezentující kompletní strukturu XML dokumentu. V této instanci je možné vyhledávat, přidávat, upravovat i mazat XML elementy a jejich atributy. [3]

Jelikož typ souboru XMI, používaný v rámci této práce, je založen na XML viz kapitola 2.3, tak je možné po definování potřebných jmenných prostorů, k němu přistupovat jako k XML dokumentu a dle toho analyzovat.

4.4 Modul `ast` pro syntaktickou analýzu rámce Django

Modul `ast` se nachází ve standardní knihovně jazyka Python od verze 2.5. Umožňuje analyzovat kód Pythonu a mít jej k dispozici ve formě abstraktního syntaktického stromu (z angl. Abstract Syntax Tree, AST), který je zpřístupněn prostřednictvím stromu složeného z objektů, jež jsou potomky třídy `ast.AST`. Tento strom je rovněž možné zkompilovat do kódového objektu Pythonu voláním vestavěné funkce `compile()`. [1]

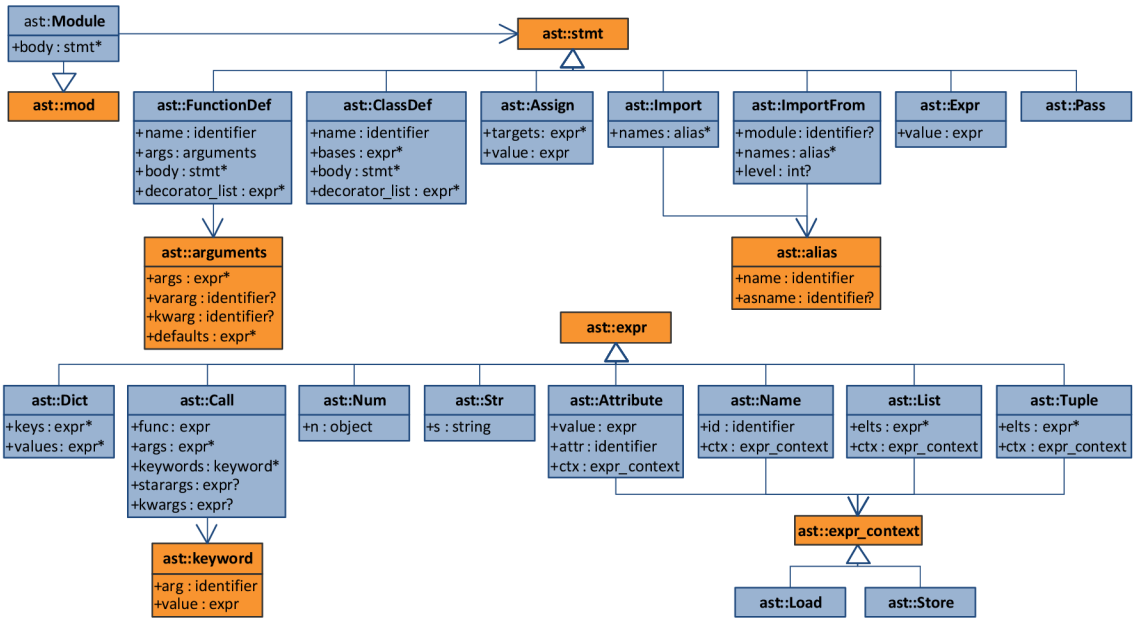
Obecně je AST používán v překladačích, kde je výstupem sémantického analyzátoru viz obrázek 4.1. AST reprezentuje derivační strom bez částí nedůležitých pro překlad, přičemž derivační strom popisuje syntaktickou strukturu vstupního řetězce tokenů. Vnitřní uzly AST zastupují operátory a následníci těchto uzlů jsou jejich operandy. [4, 16]



Obrázek 4.1: Struktura překladače [4]

Abstraktní gramatika modulu `ast` je poměrně rozsáhlá, a proto zde bude popsána jen v rámci potřeb této práce. Veškeré informace o modulu `ast` a abstraktní gramatice v této kapitole pochází z [1], kde o této problematice můžete nalézt bližší informace. Na obrázku 4.2 je vidět zjednodušený diagram tříd části abstraktní gramatiky modulu `ast`, která je použita pro syntaktickou analýzu souborů Pythonu obsahujících definice modelů backendu viz kapitola 3.2 a nastavení administrace viz kapitola 3.3. V abstraktní gramatice existuje pět vestavěných typů, a to `identifier`, `int`, `string`, `object` a `bool`. Případný otazník za názvem atributu v diagramu označuje atribut jako nepovinný, a tedy jehož hodnota může být `None`. Instance tříd nejsou vytvářeny u oranžově znázorněných tříd, od kterých dědí modře

znázorněné třídy. Celý strom si tedy lze představit jako instanci třídy `ast.Module`, která má atribut `body`, což je seznam, v němž se mohou nacházet jednotlivé prvky reprezentující funkce, třídy, přiřazení a další konstrukce jazyka Python.



Obrázek 4.2: Zjednodušený diagram tříd části abstraktní gramatiky modulu `ast` obsáhlé v rámci této práce

4.5 Aplikace South pro migraci dat rámce Django

Aby nemuselo být potřeba manuálně upravovat již vytvořené databázové tabulky pro pozměněné modely, když Django nemá vestavěnou podporu pro migraci dat, tak je možné použít některý z existujících nástrojů pro Django, např. Django Evolution, dmigrations či South, které tuto funkcionalitu mají. Poslední jmenovaný, South, je použit i v rámci aplikace této bakalářské práce. Na rozdíl od ostatních jmenovaných je vývoj South stále aktivní. Oproti dmigrations, který používá jen MySQL, je South databázově nezávislý. South podporuje databázové systémy PostgreSQL, MySQL, SQLite, Microsoft SQL Server (beta), Oracle (alpha). Při zkoušení Django Evolution i South si South na rozdíl od Django Evolution poradil s úpravou již existujících polí, kde Django Evolution nezvládl některé konverze. South dokáže dle definice modelů vytvářet, upravovat i mazat tabulky či sloupce. Při změně vlastností polí modelů zůstávají původní data v databázi zachována a rámec Django s nimi pracuje i tak pouze podle definice polí těchto modelů. Lze tak například i změnit typ pole `CharField` na `IntegerField` a zpět. Tato schopnost je velice užitečná, a je proto možné z GUI aplikace bakalářské práce libovolně měnit vlastnosti polí, namísto nutné první správné volby, která by již později nemohla být změněna. [6]

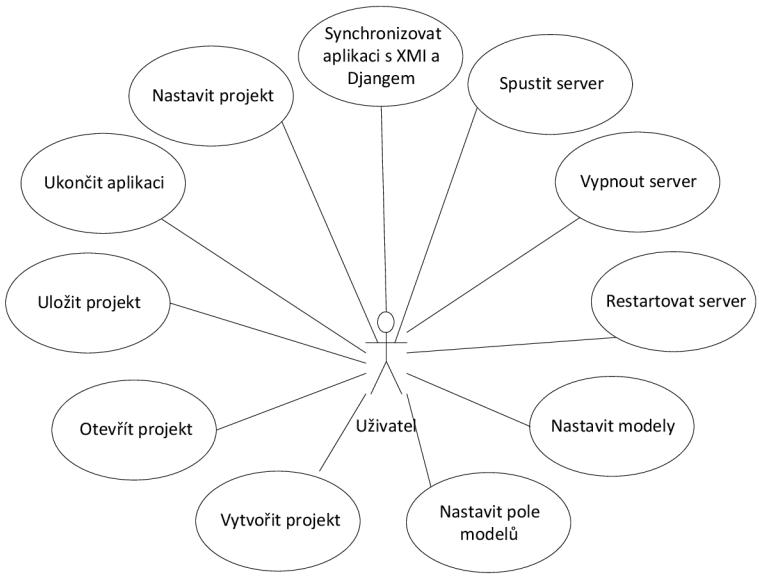
Kapitola 5

Návrh synchronizačního nástroje

Tato kapitola popisuje navržené součásti synchronizačního nástroje, bez použití názvů tříd, metod a podobně. Je zde vyobrazena koncepce grafického uživatelského rozhraní i princip činnosti nástroje. Kapitola je doplněna vyjadřovacími prostředky, jako jsou diagramy či schémata.

5.1 Případy užití nástroje

Vytvořený nástroj umožňuje uživateli provádět různé úkony, které jsou stručně znázorněny v diagramu případů užití na obrázku 5.1.



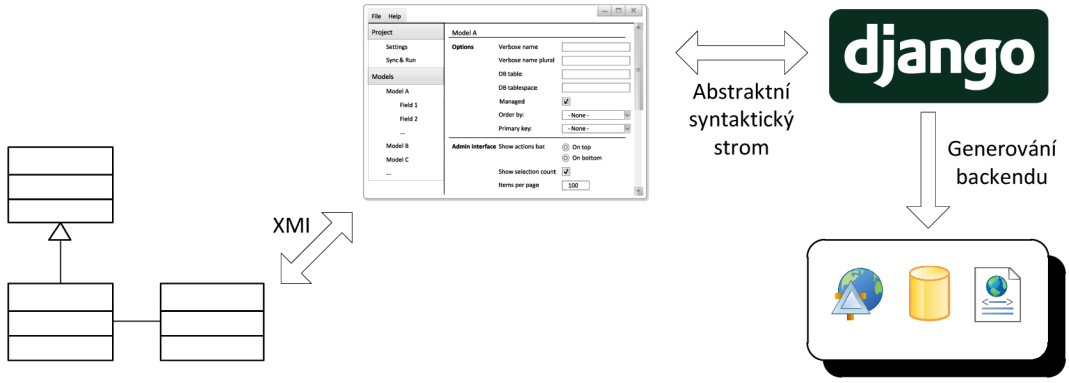
Obrázek 5.1: Diagram případů užití vytvořeného nástroje

V nástroji je možné vytvářet, otevírat a ukládat jeho projekty. Za takovýto projekt je považován adresář projektu Djanga, který je vytvořený v rámci tohoto nástroje a obsahuje potřebný konfigurační soubor. Tento soubor má příponu `.u2b` a vzniká serializací potřebných nastavení celého projektu. Možná nastavení projektu byla popsána výše v kapitole 5.2.1. Do konfiguračního souboru jsou ukládány i jednotlivé modely backendu aplikace a je-

jich pole, jejichž možná nastavení jsou uvedena v kapitolách 5.2.3 a 5.2.4. Tyto modely nástroje lze s UML modelem a backendem aplikace synchronizovat. Uložením projektu se změny zapíšou nejen do konfiguračního souboru, ale i do backendu aplikace projektu Djanga a XMI souboru propojeného s nástrojem. Pro promítnutí změn z XMI souboru či projektu Djanga do nástroje slouží v GUI tlačítka importů. Takto například importem aktuálního projektu Djanga a následným stisknutím tlačítka *Save* ve vytvořeném nástroji, jsou spolu synchronizovány XMI soubor, vytvořený nástroj i backend aplikace projektu Djanga. Pro zobrazení vygenerovaného administračního rozhraní lze přímo z grafického rozhraní nástroje spustit vývojový server Djanga, přičemž se rovnou otevře i URL adresa ve výchozím webovém prohlížeči operačního systému na definované adrese a portu, viz volby synchronizace a běhu webové aplikace v kapitole 5.2.2.

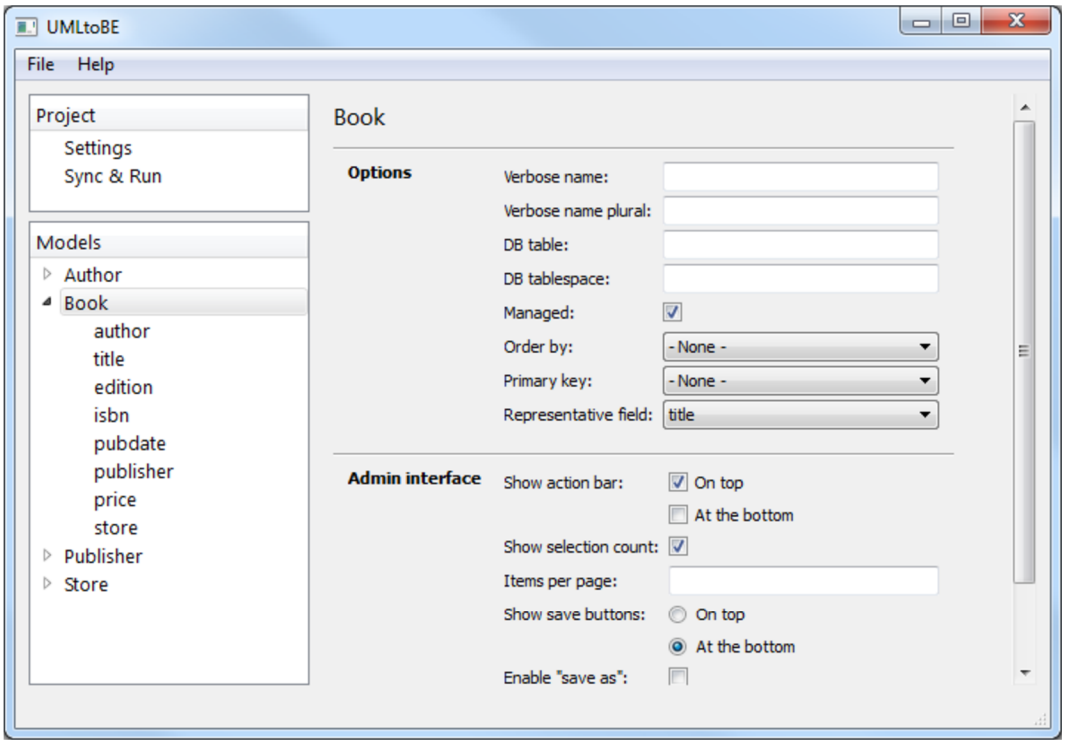
5.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní nástroje (z angl. Graphical User Interface, GUI) umožňuje importovat UML diagram tříd skrze XMI soubor, importované třídy nakonfigurovat a na jejich základě vygenerovat prostřednictvím Djanga backend aplikace. Modely a nastavení administrace Djanga nástroj generuje pomocí abstraktních syntaktických stromů. Django dle těchto definic spravuje zvolenou databázi a automaticky generuje komplexní administrační rozhraní. Kromě generování backendu aplikace z UML modelu umožňuje nástroj UML model a vygenerovaný backend aplikace synchronizovat. Je tedy možné upravovat backend aplikace nejen pomocí grafického rozhraní, ale i nadále editovat XMI soubor s UML modelem či soubory projektu Djanga a prostřednictvím tlačítek GUI spouštějících importy z obou těchto stran přenést provedené změny do vytvořeného nástroje. Na obrázku 5.2 jsou ukázány vstupy a výstupy tohoto nástroje.



Obrázek 5.2: Schéma vstupů a výstupů vytvořeného nástroje (logo Djanga převzato z [5])

Formuláře a dialogová okna jsou vytvořeny pomocí nástroje Qt Designer, z kterých je prostřednictvím nástroje pyside-uic vygenerován potřebný Python kód viz kapitola 4.2. GUI se skládá z hlavního okna, které obsahuje standardní menu nabídku, postranní panel s výběrem položek k nastavení a panel obsahující obsah pro zvolenou položku postranního panelu viz obrázek 5.3. Položky postranního panelu jsou rozděleny do kategorií *Project* a *Models*. Kategorie *Project* obsahuje položky *Settings* a *Sync & Run*. Kategorie *Models* zahrnuje všechny datové modely backendu a umožňuje patřičně nastavit je a všechny jejich pole, která jsou v panelu jejich podpoložkami.



Obrázek 5.3: Ukázka uživatelského rozhraní vytvořeného nástroje

Pro oddělení grafického uživatelského rozhraní od logické a datové části je využívána architektura MVC (Model View Controller) s použitím návrhového vzoru Observer viz [14]. Každá sekce obsahuje referenci na svůj vlastní MVC model. Jakékoli změny v jednotlivých volbách GUI jsou okamžitě předávány do jejich MVC modelů. MVC modely jsou reprezentovány jako pozorované objekty, na jejichž změně závisí pozorovatelé. Pozorovateli jsou GUI prvky viditelné uživatelem. Tímto způsobem, kdy jsou GUI prvky závislé na datech a informovány při jakékoli změně dat, je zajištěna vždy správná a aktuální reprezentace dat uživateli.

5.2.1 Sekce *Project – Settings*

V položce *Settings* lze nastavit název projektu, vybrat z nabídky typ databáze podporované Djangem a vyplnit údaje pro možnost připojení k ní. Dále lze nastavit jazyk backendu aplikace z výběru jazyků podporovaných Djangem, časové pásmo backendu aplikace a jeho administrační rozhraní. Tyto projektové volby administračního rozhraní lze přenastavit individuálně pro každý model v položkách datových modelů backendu aplikace v sekci *Models*. Možnosti těchto voleb jsou uvedeny v kapitole 5.2.3. Zmíněná volba *Date navigation by* je použita specificky pro každý model zvlášť a nelze nastavit v rámci celého projektu.

- **Name**: Název projektu
- **Database**: Databázový systém
  - **Name**: Název databáze (u SQLite cesta k souboru)
  - **User**: Uživatelské jméno (u SQLite nepoužito)

  - **Password**: Heslo (u SQLite nepoužito)
  - **Host**: Adresa databázového serveru (nevyplněno = localhost)
  - **Port**: Port databázového serveru (nevyplněno = výchozí)
  - **Initial/Auto**: Počáteční inicializace databáze/Aktualizace databáze
- **Language**: Jazyk backendu aplikace
- **Timezone**: Časová zóna backendu aplikace
- **Admin interface**: Volby administračního rozhraní viz kapitola 5.2.3

5.2.2 Sekce *Project – Sync & Run*

Položka *Sync & Run* umožňuje synchronizovat vytvořený nástroj s UML modelem a backendem aplikace a spustit vývojový server s otevřením administračního rozhraní ve webovém prohlížeči na dané IP adrese a portu. Díky synchronizaci je možné průběžně měnit datové modely, jejich nastavení či dát programátorovi webové aplikace prostor backend aplikace upravovat i přímo v programovém kódu. Tento nástroj se tak stává znovupoužitelným a neomezuje se jen na počáteční generování programového kódu. K aplikování provedených změn z XMI souboru či Django projektu do vytvořeného nástroje jsou v této části GUI připraveny tlačítka umožňující importy z obou těchto možných vstupů.

- **Sync**
  - **XMI file**: Cesta k XMI souboru obsahujícího diagram tříd
  - **Project path**: Cesta k projektu rámce Django
- **Run**
  - **IP**: IP adresa vývojového serveru
  - **Port**: Port vývojového serveru

5.2.3 Sekce *Models* – datové modely backendu

Datové modely backendu slouží k nastavení vybraných voleb Djanga, a to těch ovlivňujících výsledný backend aplikace a zároveň možných vygenerování obecně pro jakoukoli webovou aplikaci bez potřeby doprogramování pro specifické účely dané aplikace. Lze zde nastavit název modelu čitelný pro člověka v jednotném a množném čísle, název tabulky a tabulkového prostoru použitých pro model, vybrat zdali má být model připojen k již existující tabulce nebo vytvořit tabulku novou, výchozí řazení modelu, primární klíč modelu a název pole reprezentující model. Dále umožňuje nastavit administrační rozhraní modelu, stejně jako tomu bylo v kapitole 5.2.1.

- **Options**
  - **Verbose name**: Název modelu čitelný pro člověka (jedn. číslo)
  - **Verbose name plural**: Název modelu čitelný pro člověka (množ. číslo)
  - **DB table**: Název tabulky v databázi
  - **DB tablespace**: Název tabulkového prostoru v databázi

  - **Managed**: Použita pro model nová tabulka, jinak použita existující
  - **Order by**: Výchozí řazení modelu
  - **Primary key**: Primární klíč modelu
  - **Representative field**: Pole reprezentující model

- **Admin interface**
  - **Show action bar**: Umístění panelu akcí
  - **Show selection count**: Zobrazí počet vybraných položek
  - **Items per page**: Počet položek na stránce
  - **Show save buttons**: Umístění ukládacích tlačítek
  - **Enable "save as"**: Zobrazí volbu „Save as" namísto „Save and add another"
  - **Date navigation by**: Zobrazí časovou navigaci dle zvoleného pole

5.2.4 Sekce *Models* – pole datových modelů

Pole datových modelů nastavují stejně jako v kapitole 5.2.3 ty volby Djanga, které ovlivňují výsledný backend aplikace a zároveň možných vygenerování obecně pro jakoukoli webovou aplikaci bez potřeby doprogramování pro specifické účely dané aplikace. Lze zde vybrat typ pole z výběru datových a relačních typů polí, pro zvolený typ nastavit případné povinné a volitelné volby a další nastavení společné pro všechny typy polí.

- **Type**
  - **Max length**: Maximální počet znaků pole
  - **Auto now**: Nastaví poli aktuální datum a čas při uložení objektu
  - **Auto now add**: Nastaví poli aktuální datum a čas při vytvoření objektu
  - **Max digits**: Maximální počet číslic v čísle (musí být větší nežli *Decimal places*)
  - **Decimal places**: Počet desetinných míst
  - **Upload to**: Cesta pro ukládání souborů
  - **Path**: Cesta k adresáři
  - **Match**: Regulární výraz pro filtrování souborů
  - **Recursive**: Zahrne podsložky adresáře z *Path*
  - **Height field**: Název pole, kterému bude automaticky nastavena hodnota rovna šířce obrázku při uložení objektu
  - **Width field**: Název pole, kterému bude automaticky nastavena hodnota rovna výšce obrázku při uložení objektu
  - **Other model**: Název modelu, s kterým je daný model v relaci

- **Options**
  - **Null**: Ukládá prázdné hodnoty jako NULL
  - **Blank**: Povolí nechat pole nevyplněné
  - **Unique**: Určí pole jako unikátní

- **Choices**: Namísto textového pole zobrazí select box pouze s výběrem definovaných voleb
- **Default**: Výchozí hodnota pole
- **DB column**: Název sloupce v databázi
- **DB index**: Indexování pole v databázi
- **DB tablespace**: Název tabulkového prostoru v databázi
- **Editable in form**: Povolí editovat pole v administračním formuláři
- **Editable in list**: Povolí editovat pole v seznamu položek administrace
- **Help text**: Nápověda zobrazená v administračním formuláři pod daným polem
- **Show in form**: Zobrazí pole v administračním formuláři
- **Show in list**: Zobrazí pole v seznamu položek administrace
- **Show in filter**: Aktivuje filtr pro dané pole
- **Search**: Použije dané pole pro vyhledávání
- **Interface**: Nastavení zobrazeného rozhraní pro dané pole (jen pro pole typu `ManyToManyField`, `ForeignKey` a pole s nastavenou volbou *Choices*

5.3 Čtení UML diagramu tříd z XMI souboru

Nástroj umí pracovat se soubory XMI verze 2.1. Jedná se o rozšíření XML souboru s deklarovaným jmenným prostorem na URL adresu http://schema.omg.org/spec/XMI/2.1 viz kapitola 2.3. K parsování XML dokumentu je použit ElementTree XML API ze standardní knihovny jazyka Python viz kapitola 4.3. Nejprve je vstupní XMI soubor parsován pomocí ElementTree a uložen do objektu. Elementy `<packagedElement xmi:type="uml:Class">` reprezentují třídy z jazyka UML. Je tedy prováděn průchod všemi těmito elementy a pro každý z nich je vytvářen nový model. Tento element kromě atributu `xmi:type` obsahuje vždy i atribut `xmi:id` definující unikátní identifikátor dané třídy. Dále také může obsahovat další atributy či vnořené elementy. V ukázce zdrojového kódu 5.1 lze vidět příklad definice třídy v XMI, kde má třída uveden navíc atribut `name` určující její název. Uvnitř elementu třídy se mohou nacházet elementy určující atributy, operace či dědičnost třídy. Operace jsou pro možnost generování backendu aplikace nepodstatné, tak mohou být zanedbány.

```
<packagedElement xmi:type="uml:Class" xmi:id="_7f3" name="Author">
    <ownedAttribute xmi:type="uml:Property" xmi:id="_E-s" name="first_name">
        <type xmi:idref="_PJf" />
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property" xmi:id="_Hg4" name="last_name">
        <type xmi:idref="_PJf" />
    </ownedAttribute>
</packagedElement>

<packagedElement xmi:type="uml:DataType" xmi:id="_PJf" name="CharField" />
```

Zdrojový kód 5.1: Příklad definice UML třídy v XMI souboru

Elementem reprezentujícím atribut třídy je `<ownedAttribute xmi:type="uml:Property">`. Následuje tedy průchod i těmito elementy a pro každý z nich je vytvářeno nové pole modelu. Tento element obsahuje také další podstatné atributy či vnořené elementy, a to atribut `xmi:id` a atribut/vnořený element `name`

reprezentující identifikátor a název atributu třídy, které jsou poli modelu nastaveny. Dále nese informaci o svém typu skrze atribut či vnořený element `type`. Po získání typu atributu třídy je mu potřeba přiřadit vhodný typ pole modelu. Typem atributu třídy může být buď primitivní datový typ nebo reference na jinou třídu či datový typ. Je-li typem reference na třídu, tak je vytvářenému poli přiřazen typ `ForeignKey`, `ManyToManyField` či `OneToOneField` dle dané relace mezi třídami. Pakliže je typem primitivní datový typ či jiný v XMI definovaný datový typ, tak je poli patřičně dle implementovaného rozhodovacího mechanismu datový typ zvolen.

Elementem reprezentujícím dědičnost třídy je `<generalization xmi:type="uml:Generalization">`. Pokud se tento element nachází uvnitř elementu třídy, tak daná třída je potomkem třídy určené identifikátorem v atributu `general` elementu dědičnosti. Tento identifikátor je v modelu nastaven jako rodič modelu. Poté, co je dokončen průchod všemi třídami, je pro každý model nesoucí identifikátor rodiče, tento identifikátor nahrazen referencí na rodičovský model. Toto je možné provést až po dokončení průchodu všemi třídami, kdy jsou patřičně vytvořeny všechny modely.

5.4 Generování backendu aplikace prostřednictvím Djanga

Generování backendu aplikace lze rozdělit do dvou fází. Nejprve je při vytvoření projektu implementovaného synchronizačního nástroje patřičně vytvořen i projekt Djanga viz kapitola 5.4.1. Druhá fáze je provedena při uložení projektu synchronizačního nástroje viz kapitola 5.4.2, při které je generován kód backendu aplikace do souborů projektu Djanga s následným voláním potřebných příkazů pro jeho správné aplikování.

5.4.1 Inicializace projektu Djanga

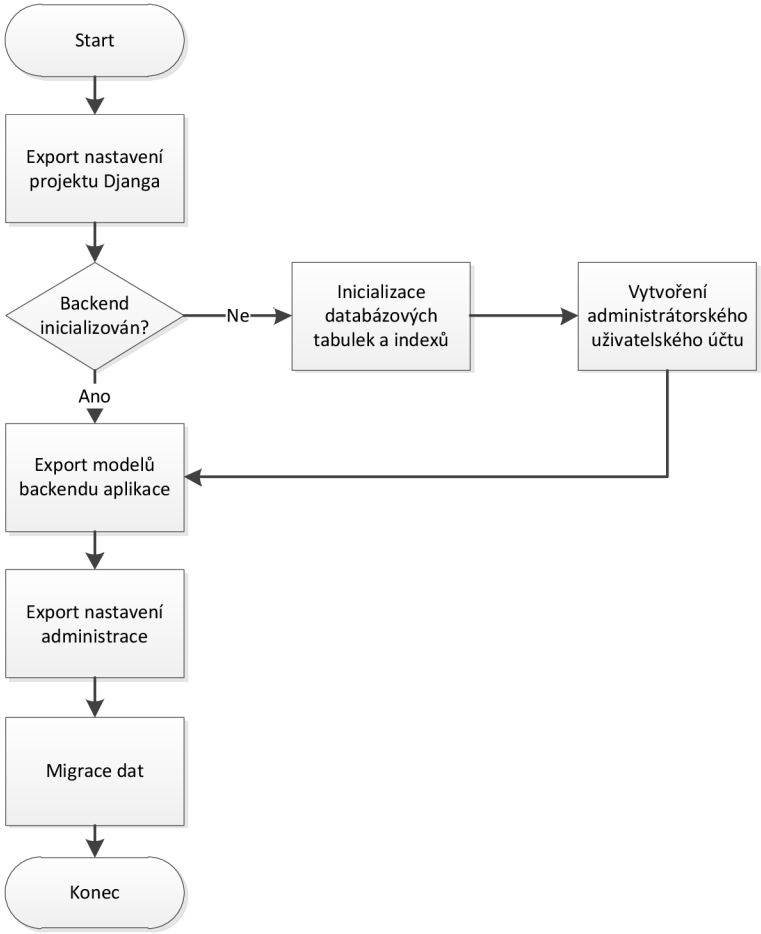
Níže jsou popsány kroky popisující inicializaci projektu Djanga včetně aplikace projektu Djanga, která bude později obsahovat nastavení a modely backendu aplikace.

1. **Vytvoření projektu Djanga**: Vytvoření projektu Djanga spočívá v pouhém spuštění skriptu `django-admin.py` s parametry pro vytvoření projektu takto: `django-admin.py startproject <název_projektu>`. Po spuštění tohoto příkazu je vytvořen adresář se jménem projektu, uvnitř kterého jsou vygenerovány soubory `__init__.py`, `manage.py`, `settings.py` a `urls.py` viz kapitola 3.1.

2. **Vytvoření aplikace Djanga**: K vytvoření aplikace Djanga je potřeba patřičně spustit vygenerovaný soubor `manage.py`, a to jako `python manage.py startapp <název_aplikace>`. Tímto je vytvořen adresář s názvem aplikace Djanga, uvnitř kterého jsou vygenerovány soubory `__init__.py`, `models.py`, `tests.py` a `views.py`. Pro potřeby generování backendu je z nich důležitý jen soubor `models.py`. Nakonec následuje nahrazení projektových souborů `settings.py` a `urls.py` připravenými soubory. Připravený soubor `settings.py` obsahuje okolo hodnot, se kterými aplikace Djanga pracuje, komentářové značky. Nástroj dokáže číst a zapisovat obsah zapsaný mezi těmito značkami. Značky jsou zapsaný ve formátu `<ZNACKA>obsah</ZNACKA>`, přičemž je zachováno odřádkování a odsazení dle potřeb jazyka Python. Dále má již tento soubor přednastavenou aplikaci South (viz kapitola 4.5) v seznamu instalovaných aplikací. Připravený soubor `urls.py` je přednastavený tak, aby bylo možné z webového prohlížeče přistupovat k administračnímu rozhraní.

5.4.2 Generování kódu do souborů projektu Django

Nyní následuje výčet kroků, které popisují, jakým způsobem je generován kód do souborů projektu Django, jenž je potřeba pro vytvoření backendu aplikace shodného s nastavením ve vytvořeném nástroji. Kroky jsou popsány na základě vývojového diagramu na obrázku 5.4, který znázorňuje algoritmus, který je použit pro generování backendu aplikace pro již vytvořený projekt.



Obrázek 5.4: Vývojový diagram generování backendu aplikace pro již vytvořený projekt

1. **Export nastavení projektu Django**: Nastavení projektu Django je uloženo v souboru `settings.py`. Zápis a čtení nastavení projektu jsou vyřešeny pomocí regulárních výrazů. Soubor `settings.py` obsahuje značky, mezi kterými je k nalezení obsah určený ke změně. Regulárním výrazem je vždy nalezena požadovaná značka a je tak možné bezproblémově změnit její obsah. Toto řešení je pro tento soubor výhodnější, nežli řešení generování kódu pro modely backendu aplikace a nastavení administrace, kde je použit modul `ast` ze standardní knihovny jazyka Python viz kapitola 4.4. Pomocí exportu regulárními výrazy nedochází ke změně formátování kódu a ztrátě komentářů, kde právě u tohoto souboru je toto velice vhodné, vzhledem k jeho předdefinované struktuře doplněné o spousty komentářů. Takto exportovány jsou výchozí nastavení databáze, časová zóna projektu, jazyk projektu, cesta k souboru `urls.py`

a je přidána vygenerovaná aplikace Djanga do seznamu instalovaných aplikací, pokud se v něm ještě nenachází.

2. **Inicializace databázových tabulek a indexů**: K inicializaci databáze je nutné spustit příkaz `python manage.py syncdb`. Ten se podívá do seznamu instalovaných aplikací a v databázi vytvoří aplikacím tabulky, které ještě nebyly vytvořeny a případné databázové indexy. Tento příkaz se běžně v Djangu používá k synchronizaci modelů s databází, avšak umí jen tabulky vytvářet a nepodporuje migraci dat. Čili v případě úpravy modelu je pak potřeba manuálně upravit i jeho tabulku v databázi. V rámci této práce je využita aplikace South, která migraci dat umožňuje viz kapitola 4.5. Pokud by nebyl nástroj `manage.py` s parametrem `syncdb` spuštěn, tak by `manage.py` neumožňoval používat příkazy pro migraci dat aplikace South.

3. **Vytvoření administrátorského uživatelského účtu**: Administrátorský uživatelský účet slouží mimo jiné k přihlášení do administračního rozhraní backendu aplikace. Pokud by po vygenerování backendu aplikace nebyl žádný takový účet k dispozici, nebylo by možné se do administrace přihlásit. Tento výchozí administrátorský účet má přihlašovací jméno `admin` a heslo `umltobe`. Po přihlášení je možné v administraci uživatelské účty libovolně spravovat a hesla měnit. Samotnému vytvoření tohoto výchozího účtu předchází kontrola, zda již v databázové tabulce uživatelů uživatel s přihlašovacím jménem `admin` neexistuje. Pokud se v ní dosud nenachází, tak je vytvořen voláním příkazu `python manage.py shell`, kde jsou tomuto příkazu zadány požadované vstupy.

4. **Export modelů backendu aplikace**: Pro generování kódu při exportu modelů backendu aplikace je použit modul `ast` ze standardní knihovny jazyka Python viz kapitola 4.4. Aby měl programátor dané webové aplikace možnost upravovat kód i manuálně a implementovat součásti neobsažené v rámci této bakalářské práce, jako jsou například operace (metody) modelů, tak není výsledný kód vždy nově generován, protože by tyto informace byly následně ztraceny. Nejprve je tedy analyzován kompletní abstraktní syntaktický strom souboru `models.py` a v něm poté prováděny patřičné úpravy dle aktuální konfigurace backendu aplikace ve vytvořeném nástroji. Z objektu AST jsou odstraněny objekty reprezentující třídy a jejich atributy, pro které již při exportu neexistují dané modely a jejich pole. Následuje průchod všemi modely a jejich poli, dle kterých jsou v objektu AST upraveny stávající třídy a vytvořeny dosud neexistující třídy.

5. **Export nastavení administrace**: Nastavení administrace je generováno, stejně jako modely, pomocí modulu `ast`. V tomto případě je nejprve analyzován soubor `admin.py` a uložen do objektu AST. Taktéž jsou obdobně prováděny operace vytváření, upravování a odstraňování objektů reprezentující třídy.

6. **Migrace dat**: K migraci dat je využit nástroj South viz kapitola 4.5. Pomocí příkazu `python manage.py schemamigration <název_aplikace> --initial` je vytvořena první migrace dané aplikace a je uložena jako skript jazyka Python do adresáře `migrations/` v cestě aplikace. Další migrace jsou vytvářeny již příkazem `python manage.py schemamigration <název_aplikace> --auto`. Vždy po vytvoření nové migrace je spuštěn příkaz `python manage.py migrate <název_aplikace>` k použití dané migrace. Ovšem ne vždy South dokáže provést migrace dat úplně automaticky

a již při vytváření migrace vypíše na standardní výstup otázku uživateli, jak chce migraci dat provést, a uživatel na standardní vstup reaguje číselnou volbou, ze zobrazené nabídky voleb. Je však možné, že takovýchto otázek na uživatele během jedné migrace následuje více, či pro některé zvolené volby je jako vstup očekáván např. Python kód. Aby nebylo nutné z vytvořeného synchronizačního nástroje spouštět průběh migrací v konzoli a vše mohlo být ucelené v rámci implementovaného grafického rozhraní, tak jsou případné požadavky při migraci delegovány do dialogového okna, kde se nachází i jedno vstupní pole, kam uživatel může psát své reakce na dané požadavky. Toto dialogové okno je tedy interaktivně propojené s příkazem pro vytváření migrací v případě, že jsou kladeny nějaké požadavky na uživatele. V situaci, kdy nelze migraci provést z důvodu, že již tabulky, které mají být migrovány, v databázi existují, tak je takováto chyba rozpoznána a migrace aplikována pomocí příkazu `python manage.py migrate <název_aplikace> --fake`, díky čemuž je zaznamenáno, že byla migrace použita, i když doopravdy spuštěna nebyla.

5.5 Synchronizace UML diagramu tříd a backendu aplikace

K synchronizaci UML diagramu tříd a backendu aplikace dochází vždy při uložení projektu vytvořeného synchronizačního nástroje. XMI soubor `models.xmi` připojený k projektu, nacházející se v cestě aplikace, je vždy při uložení patřičně pozměněn, a to opět s použitím ElementTree API jako při importu viz kapitola 5.3. Obdobně je při uložení proveden export aktuálního nastavení backendu do projektu Django prostřednictvím abstraktního syntaktického stromu, jak bylo již popsáno v kapitole 5.4.

Pokud uživatel provede úpravy v XMI souboru či Django projektu a přeje si vše sesynchronizovat na jejich základě, může použít tlačítka importů v sekci *Project – Sync & Run* viz kapitola 5.2.2. Tím jsou změny provedené v XMI souboru/Django projektu přeneseny do vytvořeného nástroje a následným uložením šířeny jak do XMI souboru, tak do Django projektu. Tímto způsobem je možné synchronizovat UML model, vytvořený nástroj a backend aplikace.

Vytvořený synchronizační nástroj je tedy jakýsi prostředník mezi UML modelem a backendem aplikace. Aby mohl při importu rozpoznat, které modely či pole modelů nebyly z XMI/Django importovány, je na začátku nastaven čas zahájení importu. Modely i jejich pole nesou informaci o datu jejich vytvoření a jejich poslední aktualizaci. Čas poslední aktualizace je nastavován vždy při importu. Pokud po skončení importu se u některého modelu/pole neshoduje čas poslední aktualizace s časem zahájení importu, tak daný model/pole má být odstraněn.

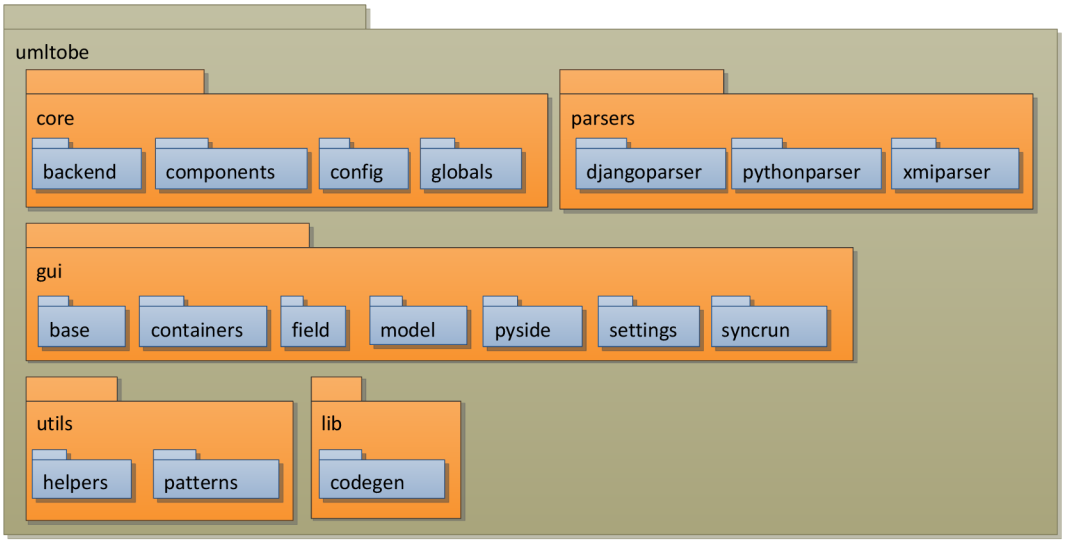
Kapitola 6

Implementace

V této kapitole jsou popsány implementační detaily a architektura samotné aplikace až na úroveň jednotlivých tříd.

6.1 Architektura synchronizačního nástroje

Na obrázku 6.1 lze vidět organizaci balíků a modulů vytvořeného nástroje. Moduly jsou v jazyce Python soubory, které stejně jako balíky definují jmenný prostor pro svůj obsah.



Obrázek 6.1: Organizace balíků a modulů vytvořeného nástroje

Balík nejvyšší úrovně se nazývá `umltobe` a je tvořen pěti hlavními balíky `core`, `gui`, `parsers`, `utils` a `lib`. Ty jsou dále složeny z jednotlivých modulů. Následuje stručný popis všech balíků a modulů implementovaných ve vytvořeném nástroji.

- `core`: Jádro nástroje, obsahuje veškerá uložená data, s kterými nástroj pracuje
  - `backend`: Modul definující backend aplikace a možnost jeho importu
  - `components`: Modul obsahující komponenty, z kterých je backend aplikace složen
  - `config`: Modul mající na starost konfiguraci synchronizačního nástroje

  - `globals`: Modul obsahující globální konstanty jako jsou časové zóny, výčet možných jazyků backendu aplikace apod.

- `gui`: Obsahuje moduly grafického uživatelského rozhraní
  - `pyside`: V tomto případě se nejedná o modul, ale vnořený balík obsahující prvky grafického uživatelského rozhraní, vytvořené nástrojem Qt Designer a převedené do souborů Pythonu nástrojem pyside-uic
  - `base`: Modul umožňující spuštění grafického rozhraní a hlavní okno aplikace
  - `containers`: Modul obsahující grafické kontejnery, jež jsou použity pro konkrétní grafické prvky
  - `field`: Modul sekce *Models* – pole datových modelů viz kapitola 5.2.4
  - `model`: Modul sekce *Models* – datové modely backendu viz kapitola 5.2.3
  - `settings`: Modul sekce *Project* – *Settings* viz kapitola 5.2.1
  - `syncrun`: Modul sekce *Project* – *Sync & Run* viz kapitola 5.2.2

- `parsers`: Balík zahrnující syntaktické analyzátory
  - `djangoparser`: Modul mající na starost importy a exporty Djanga
  - `pythonparser`: Modul pro syntaktickou analýzu jazyka Python
  - `xmiparser`: Modul mající na starost importy a exporty XMI souboru

- `utils`: Balík obsahující moduly s užitečnými pomůckami
  - `helpers`: Modul obsahující pomocné funkce
  - `patterns`: Modul obsahující návrhové vzory

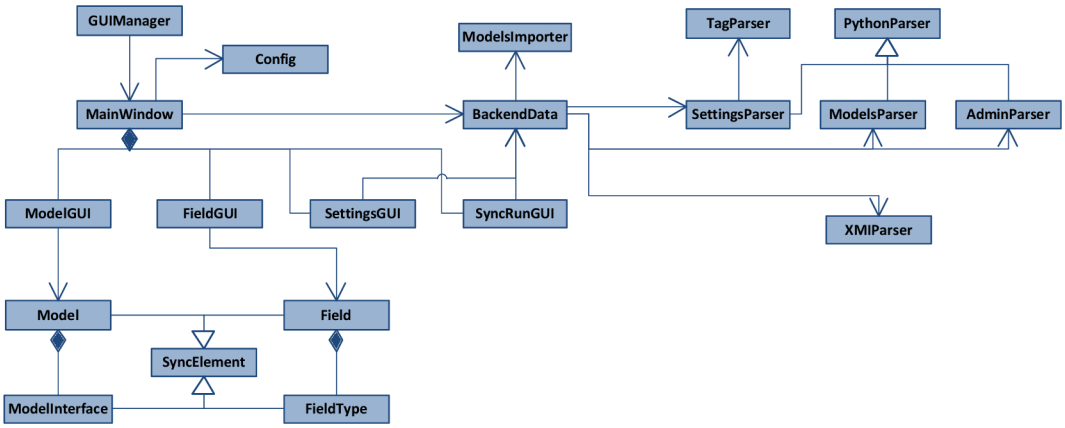
- `lib`: Balík, který je určen pro externí knihovny
  - `codegen`: Modul šířený pod licencí BSD, vytvořen roku 2008 a jehož autorem je Armin Ronacher. Umožňuje generovat AST do kódu Pythonu. Byl lehce opraven, aby vyhovoval potřebám vytvořeného nástroje s aktuální specifikací modulu `ast` jazyka Python

V balíku `umltobe` se také nachází adresář `data`, který obsahuje připravené soubory `settings.py` a `urls.py`, jenž při vytvoření projektu Djanga nahrazují Djangem vygenerované stejnojmenné soubory, jak bylo popsáno v kapitole 5.4. Také je v této složce umístěn konfigurační soubor vytvořeného nástroje nazývající se `umltobe.cfg`, jenž je tvořen serializací potřebných vlastností nástroje, které mají být uloženy. Serializace je prováděna prostřednictvím modulu `shelve` ze standardní knihovny jazyka Python, který umožňuje serializovat datový typ `dict` – slovník, také znám jako asociativní pole.

Nyní budou popsány nejvýznamnější implementované třídy. Třídy jsou seskupovány v modulech uvedených výše. Pro oddělení grafického uživatelského rozhraní od logické a datové části aplikace je využita architektura MVC (Model-View-Controller). V balíku `patterns` jsou vytvořeny třídy `Observer` a `Observable` dle návrhového vzoru Observer viz [14]. MVC modely jsou potomky třídy `Observable` a třídy uživatelského rozhraní implementují rozhraní třídy `Observer`. V případě třídy `Observer` se jedná z hlediska objektově orientovaného programování o rozhraní, nehledě na to, že jazyk Python rozhraní přímo

vytvářet neumožňuje, ale v případě, že v potomku třídy `Observer` není implementována jediná jeho metoda `update()`, tak je vyvolána výjimka `NotImplementedError`.

Ve zjednodušeném diagramu tříd na obrázku 6.2 lze vidět implementované třídy a jejich vzájemné relace. Diagram pro přehlednost neukazuje všechny třídy a relace, ale jen ty nejpodstatnější.



Obrázek 6.2: Zjednodušený diagram tříd vytvořeného nástroje

Stručný popis uvedených tříd je:

- `GUIManager`: Správce grafického uživatelského rozhraní
- `MainWindow`: Hlavní okno aplikace, jehož MVC modelem je instance třídy `BackendData`
- `Config`: Třída umožňující číst a zapisovat konfiguraci aplikace
- `BackendData`: Třída obsahující samotný backend aplikace
- `ModelsImporter`: Třída provádějící import backendu aplikace do nástroje z XMI či Djanga
- `ModelGUI`: Qt widget sekce *Models* – datové modely backendu viz kapitola 5.2.3, jehož MVC modelem je instance třídy `Model`
- `FieldGUI`: Qt widget sekce *Models* – pole datových modelů viz kapitola 5.2.4, jehož MVC modelem je instance třídy `Field`
- `SettingsGUI`: Qt widget sekce *Project – Settings* viz kapitola 5.2.1, jehož MVC modelem je instance třídy `BackendData`
- `SyncRunGUI`: Qt widget sekce *Project – Sync & Run* viz kapitola 5.2.2, jehož MVC modelem je instace třdy `BackendData`
- `SyncElement`: Třída definující společné vlastnosti pro potomky, jež mají být synchronizovány
- `Model`: Potomek třídy `SyncElement`, model backendu aplikace

- `ModelInterface`: Součást třídy `Model`, volby administračního rozhraní modelu
- `Field`: Potomek třídy `SyncElement`, pole modelu
- `FieldType`: Součást třídy `Field`, typ a typově specifické volby pole modelu
- `XMIParser`: Syntaktický analyzátor umožňující čtení a zápis UML diagramu tříd v XMI souboru
- `PythonParser`: Syntaktický analyzátor umožňující čtení a zápis abstraktního syntaktického stromu modulu `ast` ze standardní knihovny jazyka Python
- `SettingsParser`: Syntaktický analyzátor umožňující čtení a zápis souboru `settings.py` projektu Django, potomek třídy `PythonParser`
- `TagParser`: Syntaktický analyzátor umožňující čtení a zápis hodnot umístěných mezi komentářovými značkami v souboru `settings.py` projektu Django, využívaný v třídě `SettingsParser`
- `ModelsParser`: Syntaktický analyzátor umožňující čtení a zápis souboru `models.py` projektu Django, potomek třídy `PythonParser`
- `AdminParser`: Syntaktický analyzátor umožňující čtení a zápis souboru `admin.py` projektu Django, potomek třídy `PythonParser`

6.2 Konvence dodržované při implementaci

Při implementaci byla snaha dodržet konvence jazyka Python dle PEP 8 viz [15]. Avšak v modulech balíku `gui` nejsou názvy proměnných a metod psány malým písmem s oddělením slov pomocí podtržítka, jak je uvedeno v PEP 8 a použito i v ostatních balících vytvořeného nástroje. V modulech balíku `gui` je pro konzistenci s použitým rámcem Qt prostřednictvím knihovny PySide, použit styl CamelCase s počátečním malým písmenem. Další rozdíl od PEP 8 je nedodržení limitu maximálně 79 znaků na řádek, kde obzvlášť v modulech balíku `gui` by to mohlo být na úkor přehlednosti. Bylo lepší volit delší názvy vzhledem k počtu tolika ovládacích prvků, aby byl napsaný kód srozumitelnější.

Moduly, jakožto soubory definující jmenný prostor, obsahují i více tříd. Což je v Pythonu zvykem na rozdíl např. od jazyku Java, kde je konvencí umísťovat každou třídu zvlášť do souboru. Takzvané „gettery" a „settery" nejsou pro atributy tříd vytvořeny. V případě, že je potřeba při získání či nastavení atributu provést nějaké další úkony, tak jsou danému atributu pomocí vestavěné funkce `property()` „getter" a „setter" funkce přiřazeny a jsou posléze volány automaticky vždy při získání/nastavení hodnoty atributu.

Kapitola 7

Reálné použití a testování

Tato kapitola se zaměřuje na reálné použití a testování vytvořeného synchronizačního nástroje. Nejprve je v kapitole 7.1 vybrán vhodný UML nástroj a poté v kapitole 7.2 otestována na jednoduchém příkladu funkčnost vytvořeného nástroje.

7.1 Výběr vhodného UML nástroje

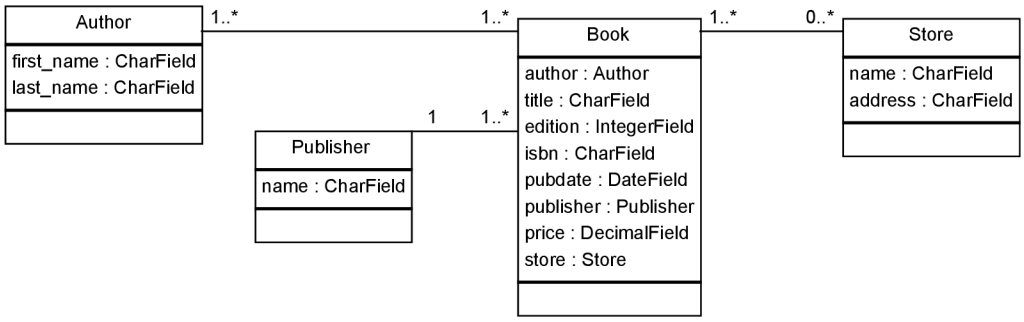
Vzhledem k tomu, že vstupem je UML diagram tříd, je potřeba vybrat nějaký UML nástroj schopný uložit diagram tříd do formátu XMI 2.1, s kterým vytvořený synchronizační nástroj pracuje. Stručné uvedení do UML nástrojů schopných s tímto typem souboru pracovat byl popsán v kapitole 2.2. Nejvhodnější volbou pro potřeby této ukázky bude výběr takového UML nástroje, který je k dispozici zdarma. S prozkoumaných nástrojů jsou to tedy Modelio 2.1.1, ArgoUML 0.34 pre-alpha UML2 a Software Ideas Modeler 5.20. Všechny tyto nástroje umožňují exportovat UML diagram tříd do XMI 2.1 formátu.

Pro testování vytvořeného synchronizačního nástroje byl nakonec zvolen ArgoUML 0.34 pre-alpha UML2. Oproti nástroji Modelio je s ním lépe možné zobrazit zpět importovaný XMI soubor do diagramu. Nástroj Software Ideas Modeler také není špatnou volbou, avšak produkuje XMI soubory ne zcela kompatibilní s ostatními nástroji, a to včetně i např. známého komerčního nástroje Enterprise Architect od firmy Sparx Systems. Bohužel ArgoUML již není vyvíjen a XMI 2.1 podporuje jen ve své pre-alpha verzi. Pro účely této bakalářské práce však plně dostačuje a i veškeré nezdokumentované testy probíhaly převážně s ním. Přestože se jedná pouze o pre-alpha verzi, co se týče diagramu tříd, nevyskytly se po dobu práce s tímto nástrojem žádné problémy. Tato pre-alpha verze se nachází standardně v cestě instalace ArgoUML 0.34, kde jsou k dispozici skripty `argouml2.bat` pro spuštění pre-alpha verze na Windows a `argouml2.sh` spouštějící pre-alpha verzi v operačním systému GNU-/Linux. Při jeho spouštění je zobrazeno upozornění „You are running an experimental version not meant for productive work!". Avšak jak již bylo napsáno výše, pro potřeby této práce tento nástroj plně dostačuje a z ostatních zdarma dostupných nástrojů byla práce s ArgoUML nejpohodlnější.

7.2 Test synchronizačního nástroje na jednoduchém diagramu tříd popisujícího knihkupectví

Funkčnost diagramu tříd bude otestována na jednoduchém diagramu tříd popisujícího knihkupectví. Digram bude složen celkem ze čtyř vzájemně propojených tříd. Kniha bude mít

jednoho či více autorů a autor bude mít napsaných jednu či více knih. Každá kniha bude mít právě jednoho vydavatele a vydavatel bude moci mít vydaných jednu a více knih. Knihy budou moci být k dispozici v prodejnách modelovaného knihkupectví. Na obrázku 7.1 je vidět vytvořený diagram tříd.

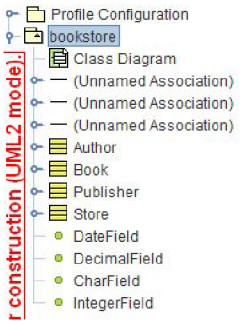


Obrázek 7.1: Jednoduchý diagram tříd knihkupectví

Dle vztahů mezi třídami bude při importu rozpoznáno, jaký typ relačního pole má být zvolen pro Django. Název relačního pole je odvozen od názvu atributu třídy, který má jako datový typ uveden danou třídu v relaci. Ostatní datové typy jednotlivých atributů jsou rovnou pojmenovány tak, jak je zná Django. V případě, že by byly uvedeny jiné datové typy (např. primitivní typy jazyka UML), tak jim dle implementovaného rozpoznání bude typ pole patřičně přiřazen.

Vytvořený diagram tříd je z *Menu–File–Export XMI* nástroje ArgoUML exportován do XMI souboru. V synchronizačním nástroji je posléze vytvořen nový projekt na základě tohoto XMI souboru. Už v tuto chvíli je v případě uložení projektu backend aplikace automaticky vygenerován a je možné zobrazit administrační rozhraní z webového prohlížeče na zvolené IP adrese a portu.

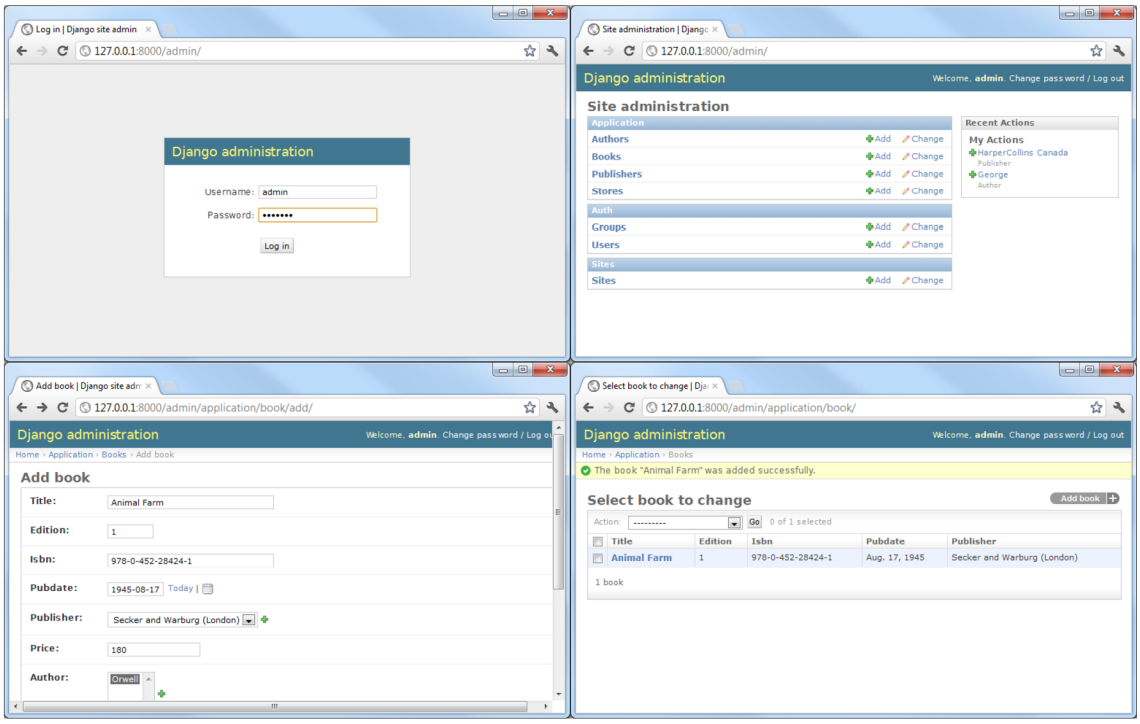
V případě úpravy importovaných modelů a následného uložení je pozměněn i importovaný XMI soubor. Přesně tedy ne přímo zadaný soubor, ale jeho kopie v cestě vygenerované aplikace projektu Djanga. Do nástroje ArgoUML může být XMI soubor importován skrze *Menu–File–Import XMI*. Takto však není automaticky vytvořen diagram, ale pouze načten UML model viz obrázek 7.2, pro něhož je možné diagramy vytvářet.



Obrázek 7.2: UML model zobrazený v nástroji ArgoUML

Pro vytvoření diagramu tříd z importovaného UML modelu stačí kliknout pravým tlačítkem myši na položku *Class Diagram* v daném UML modelu a z kontextového menu zvolit volbu *Add All Classes in Namespace*. Tím jsou do diagramu tříd přidány všechny třídy z tohoto UML modelu. Aby byly v diagramu tyto třídy přehledně uspořádané, je možné je nechat vyrovnat automaticky přes *Menu–Arrange–Layout* či si je porovnat manuálně přetáhnutím.

Nehledě na různá nastavení, která je možná backendu aplikace nastavit, je vhodné pro každý model backendu zvolit alespoň pole, které bude model reprezentovat (volba *Representative field*). Například u modelu autora je tedy užitečné zvolit jako reprezentativní pole jeho příjmení namísto jména. Na obrázku 7.3 lze vidět několik screenshotů vygenerované administrace.



Obrázek 7.3: Screenshoty vygenerované administrace

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat nástroj umožňující obousměrnou synchronizaci UML diagramu tříd a backendu aplikace kdykoliv v průběhu tvorby aplikace. Vzhledem k tomu, že měl synchronizační nástroj umět pracovat s UML diagramem tříd, bylo potřeba prozkoumat nástroje umožňující tvorbu UML diagramů tříd a používané formáty souborů pro ukládání těchto diagramů. Dle zjištěných informací podporovala spousta nástrojů jeden společný datový formát, kterým byl formát XMI (XML Metadata Interchange). XMI je standardizován konsorciem OMG (Object Management Group) a umožňuje výměnu UML modelů mezi různými nástroji. Od svého vzniku však byl XMI standard několikrát pozměněn a jeho nové verze již nebyly zpětně kompatibilní s verzemi předchozími. Různé UML nástroje často podporují odlišné verze XMI standardu a navíc ne vždy je implementují přesně, jak byly standardizovány, čímž není možné pokaždé zaručit jejich přenositelnost mezi různými nástroji. Verze XMI standardu 2.1, jenž je podporována v synchronizačním nástroji, byla vybrána s ohledem na její aktuálnost a početnou podporu mezi odlišnými nástroji. K vytvoření backendu aplikace na základě UML modelu byl využit webový aplikační rámec Django, napsaný v jazyce Python, a aplikace South, umožňující Djangu provádět migrace dat. Django mimo jiné umožňuje definovat datový model aplikace, dle něhož a potřebných konfigurací, je schopen automaticky vygenerovat komplexní administrační rozhraní. Pro čtení a zápis programového kódu Djanga využívá synchronizační nástroj syntaktické analýzy abstraktních syntaktických stromů potřebných souborů Djanga.

Vytvořený synchronizační nástroj umožňuje prostřednictvím grafického uživatelského rozhraní backend aplikace různorodě parametrizovat. Cílem však nebylo vytvořit plnohodnotné grafické rozhraní pro Django, nýbrž využít Django jako prostředek pro vznik a správu backendu aplikace. Programátorovi, který využije pro tvorbu webové aplikace schopnosti tohoto nástroje, je ponechána možnost frontend aplikace manuálně naprogramovat, případně i backend aplikace přímo v programovém kódu upravovat. Avšak díky tomuto nástroji může backend aplikace stvořit na základě UML diagramu tříd, který zpravidla stejně bývá využíván pro návrh aplikací, a ušetřit si tak čas při implementaci aplikace. Vzhledem k tomu, že vytvořený nástroj umožňuje nadále synchronizovat UML diagram tříd i backend aplikace, tak zůstává i UML diagram tříd po dobu tvorby backendu aplikace stále aktuální.

V rámci testování synchronizačního nástroje byla úspěšně ověřena jeho schopnost na základě vstupního UML diagramu tříd vygenerovat prostřednictvím Djanga backend aplikace. Stejně tak byly otestovány možnosti synchronizace a úpravy již vygenerovaného backendu ať už změnou UML diagramu tříd, parametrizací v grafickém rozhraní nástroje či manuálním programováním patřičných souborů Djanga.

V nástroji jsou však dosud ne zcela vyřešeny nějaké problémy. Vzhledem k množství

ovládacích prvků, nejsou ošetřeny všechny možné vstupy a předpokládá se, že uživatel bude do nástroje zapisovat pouze správné hodnoty stejně tak, jako kdyby přímo psal kód aplikace v Djangu. Taktéž nejsou stále ošetřeny některé v Djangu zakázané kombinace a je možné při nesprávné konfiguraci modelů znemožnit vygenerování backendu aplikace či přístup do administračního rozhraní. Vytvořený nástroj by vzhledem k použitým technologiím měl být multiplatformní, avšak testován byl primárně v prostředí Windows a není v tuto chvíli zaručeno, že je vše plně funkční i v prostředí Linux/Unix/Mac OS.

Při možném pokračováním projektu by bylo možné vytvořený nástroj povýšit až na úroveň plnohodnotného grafického rozhraní, umožňující backend aplikace vytvářet a spravovat i bez potřeby vstupního UML diagramu tříd. Co se diagramu tříd týče, mohly by být implementovány vazby i na zbylé prvky diagramu tříd, jako např. výčtový typ, rozhraní, abstraktní třídy a podobně. Podpora XMI 2.1 by se dala rozšířit i na ostatní verze XMI. Kromě práce s XMI soubory by nástroj mohl umožňovat přímo UML diagram tříd kreslit. Dále by nástroj mohl pracovat i s jinými typy diagramů, kupříkladu s ER diagramem. Lze také do nástroje implementovat schopnost pracovat s více vlastnostmi Djanga, vzhledem k jeho robustnosti.

Jak lze vidět, možností rozšíření jsou spousty. V případě, že by se do pokračování projektu vložili například i někteří vývojáři z komunity, která se pohybuje okolo Djanga, mohl by se v brzké době z tohoto nástroje stát velice dobrý pomocník pro tvorbu webových aplikací.

Literatura

[1] *Abstract Syntax Trees* [online]. Last updated on May 01, 2012 [cit. 2012-05-01]. Dostupné na: <http://docs.python.org/library/ast.html>.

[2] *Django 1.3 documentation* [online]. (c) 2005–2012 [cit. 2012-04-25]. Dostupné na: <https://docs.djangoproject.com/en/1.3/>.

[3] *The ElementTree XML API* [online]. Last updated on Feb 23, 2012 [cit. 2012-04-09]. Dostupné na: <http://docs.python.org/library/xml.etree.elementtree.html>.

[4] *Formální jazyky a překladače: Úvod do překladačů* [online]. Brno: FIT VUT v Brně, 2011-09-19 [cit. 2012-05-01]. Podklady k přednáškám, Kapitola II. Dostupné na: <https://www.fit.vutbr.cz/study/courses/IFJ/private/prednesy/Ifj02-cz.pdf>.

[5] *Official Django Logos* [online]. (c) 2005–2012 [cit. 2012-05-14]. Dostupné na: <https://www.djangoproject.com/community/logos/>.

[6] *South* [online]. Last modified 05/08/12 11:39:01 [cit. 2012-05-09]. Dostupné na: <http://south.aeracode.org/>.

[7] *MOF 2.0/XMI Mapping Specification, v2.1* [online]. [b.m.]: Object Management Group (OMG), 2005 [cit. 2012-04-24]. Formal/05-09-01. Dostupné na: <http://www.omg.org/spec/XMI/2.1/PDF/>.

[8] *OMG Unified Modeling Language (OMG UML), Infrastructure, v2.3* [online]. [b.m.]: Object Management Group (OMG), 2010 [cit. 2012-04-24]. Formal/2010-05-03. Dostupné na: <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF>.

[9] *OMG Unified Modeling Language (OMG UML), Superstructure, v2.3* [online]. [b.m.]: Object Management Group (OMG), 2010 [cit. 2012-04-24]. Formal/2010-05-05. Dostupné na: <http://www.omg.org/spec/UML/2.3/Superstructure/PDF>.

[10] Arlow, J. a Neustadt, I. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 1. vyd. Brno: Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.

[11] Grose, T. J., Doney, G. C. a Brodsky, S. A. *Mastering XMI: Java programming with XMI, XML, and UML.* New York: John Wiley, 2002. 434 s. ISBN 0-471-38429-1.

[12] Holovaty, A. a Kaplan-Moss, J. *The definitive guide to Django: Web development done right.* 2. vyd. New York: Springer-Verlag, 2009. 499 s. ISBN 978-1-4302-1937-8.

[13] Křena, B. a Kočí, R. *Úvod do softwarového inženýrství, studijní opora.* Brno: FIT VUT v Brně, 2010.

[14] Pavlíček, L. *Návrhové vzory (design paterns)* [online]. 02.02.2008, 11:01 [cit. 2012-05-08]. Dostupné na: <`http://objekty.vse.cz/Objekty/Vzory`>.

[15] Rossum, G. van a Warsaw, B. *Style Guide for Python Code* [online]. [b.m.]: Python Software Foundation, 2001, Last-Modified: 2012-04-28 00:51:37 [cit. 2012-05-13]. PEP, 8. Dostupné na: <`http://www.python.org/dev/peps/pep-0008/`>.

[16] Češka, M., Hruška, T. a Beneš, M. *Překladače, skriptum VUT Brno.* 1. vyd. Brno: Ediční středisko VUT Brno, 1993. 262 s. ISBN 80-214-0491-4.

Příloha A

Obsah CD

Součástí bakalářské práce je CD, které obsahuje všechny zdrojové soubory implementovaného synchronizačního nástroje a příklady UML modelů uložených ve formátu XMI. Dále je na CD umístěna elektronická verze bakalářské práce ve formátu PDF i její zdrojové soubory systému LaTeX.

Příloha B

Instalace nástroje

Vzhledem k tomu, že je nástroj napsán v jazyce Python verze 2.7, budete potřebovat stáhnout a nainstalovat interpret jazyka Python, který je k nalezení na adrese http://python.org/download/. Také je vhodné přidat cestu k interpretu Pythonu a cestu se skripty (uvedené níže) do proměnné PATH operačního systému. Je to potřeba i pro správné provedení následujících kroků (např. pro spuštění `easy_install` a `pip`).

- Windows: `C:\Python27\Scripts\`
- Linux/Unix: `/usr/local/bin/`

(cesty mohou být i odlišné dle instalace Pythonu a operačního systému)

Pozn.: *Vzhledem k použitým technologiím by nástroj měl být multiplatformní, avšak byl testován převážně v prostředí Windows.*

Kroky instalace:

1. Stáhnout a nainstalovat `setuptools`:
   http://pypi.python.org/pypi/setuptools#files
   - Windows: stáhnout a spustit `setuptools-0.6c11.win32-py2.7.exe`
   - Linux/Unix: stáhnout a rozbalit `setuptools-0.6c11.tar.gz`
     z adresáře rozbaleného archivu: `sudo python setup.py install`
2. Nainstalovat `pip`:
   - Windows: `easy_install pip`
   - Linux/Unix: `sudo easy_install pip`
3. Nainstalovat Django verze 1.3 (v průběhu této bakalářské práce byla vydána i verze 1.4, avšak s tou již nebyl nástroj testován):
   - Windows: `pip install django==1.3`
   - Linux/Unix: `sudo pip install django==1.3`
4. Nainstalovat South
   - Windows: `pip install south`

- Linux/Unix: `sudo pip install south`

5. Stáhnout a nainstalovat PySide – `PySide_Binaries_` dle operačního systému: http://qt-project.org/wiki/PySideDownloads
6. Rozbalit na lokální disk archiv `UMLtoBE.tar.gz` přiložený na CD
   - Windows: z adresáře rozbaleného archivu: `python setup.py install`
   - Linux/Unix: z adresáře rozbaleného archivu: `sudo python setup.py install`

Spuštění nástroje:
Skript spouštějící nástroj se nachází ve skriptech v cestě instalace Pythonu

- Windows: `C:\Python27\Scripts\umltobe_gui.py`
- Linux/Unix: `sudo /usr/local/bin/umltobe_gui.py`

(cesty mohou být i odlišné dle instalace Pythonu a operačního systému)

Rozšiřující knihovní závislosti:

- V případě potřeby používání polí typu `ImageField` je nutné nainstalovat Python Imaging Library (PIL):
  - Windows: http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe
  - Linux/Unix: stáhnout a rozbalit http://effbot.org/downloads/Imaging-1.1.7.tar.gz
    z adresáře rozbaleného archivu: `sudo python setup.py install`
- Databáze SQLite3 funguje již v základu. Avšak pokud použijete jiný databázový systém a nebude možné se připojit, je potřeba doinstalovat potřebné knihovny pro možnost připojení se k databázi z jazyka Python viz https://docs.djangoproject.com/en/1.3/topics/install/#database-installation.