TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Adaptační mechanismus pro datamining a QA testování

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Miroslav Němec**
Vedoucí práce: Ing. Jiří Jeníček, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Adaptation mechanism for datamining and QA testing

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Miroslav Němec**
Supervisor: Ing. Jiří Jeníček, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Adaptační mechanismus pro datamining a QA testování

Jméno a příjmení: **Bc. Miroslav Němec**
Osobní číslo: M17000133
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Student zmapuje možnosti využití a užití jednotlivých postupů při práci s databází (Entity Framework, MySQL, Telerik Open Access), aplikací Excel (interop, EPPlus) a interně vyvíjeným testovacím nástrojem (Selenium, FF, WinfoTaskManager)
2. Navrhne a vytvoří aplikaci pro transformaci, modifikaci a editování dat včetně souvisejících pomocných nástrojů podle požadavků zadaných z winfo s.r.o. s důrazem na optimalizaci celého procesu
3. Otestuje funkčnost editačního nástroje pro transformační funkce, exportování a importování dat, kalendářní posuny s ohledem na omezující podmínky, parametrizovanou randomizaci a validitu transformace dat
4. Sepíše dokumentaci k aplikaci

Rozsah grafických prací:	Dle potřeby dokumentace
Rozsah pracovní zprávy:	40-50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Steve McConnell: Dokonalý kód, Computer Press, a.s., 2006.
[2] Olivia Parr Rud: Datamining, Computer Press, a.s., 2006.
[3] Jeffrey Friedl: Mastering Regular Expressions, O'Reilly Media; 2006
[4] Jason Roberts: Clean C#
[5] Dokumentace k winfoBot (interní dokument)

Vedoucí práce:	Ing. Jiří Jeníček, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	9. října 2019
Předpokládaný termín odevzdání:	18. května 2020

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2019

Prohlášení

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

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

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

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 1.6.2020

Podpis: Bc. Miroslav Němec

Abstrakt

Tato diplomová práce se zaměřuje na problematiku editace, transformace, adaptace a verifikace velkých datových sad z velkého počtu různých zdrojů za účelem automatizace procesů zpracovávajících data z a pro data mining. Nastiňuje využití vlastních výpočtových funkcí pro transformační procesy a jednotlivých řešení pro adaptační procesy. Výstupem z práce je funkční .NET aplikace.

Klíčová slova: .NET, automatizace, verifikace, adaptace, transformace

Abstract

This diploma thesis is focused on editation, tranformation, adaptation and verification problems of big data collected from numerous sources with the purpose of data processing automatization for and from data mining.It drafts the usage of self-implemented calculation functions for transformation process and particular solutions for adaptation process. The result of the thesis is a functional .NET application.

Keywords: .NET, automatization, verification, adaptation, transformation

Poděkování

Mé díky patří zadavatelské firmě, jež mi umožnila vstup do světa analýzy webových dat a reálného světa programování. Taktéž patří vedoucímu této práce doktoru Jiřímu Jeníčkovi za neobvyklý pohled na svět dospělých a jeho čas a cenné rady.

Obsah

Seznam obrázků

Seznam zkratek

DB	Databáze
Group	Dialog_question_group
Option	Dialog_option
GUI	Graphical user interfaces

Úvod

Tato diplomová práce vznikala ve firmě zaměřující se na zpracování a analýzu webových dat, konkrétně na analýzu pojištění automobilů a motocyklů, například pro novinové články či internetové porovnávače. Firma posledních několik let investuje velké úsilí do automatizace veškerých procesů. Vzhledem k rozsáhlosti dat a jejich struktur je však v první řadě důležité pochopit jednotlivé souvislosti a zákonitosti, kterými se tato data řídí. I přes snahu unifikovat vstupní data, jež pocházejí z desítek různých zdrojů a slouží k mnoha různým účelům, jsou databázové struktury rozsáhlé a kvůli nestálosti dat je třeba co možná nejvíce autonomního zpracování. Veškerý vývoj automatizačních softwarů probíhá v plném nasazení při vykonávání běžné činnosti firmy. Vývoj aplikací proto postupuje minoritními kroky vpřed tak, aby integrace nových součástí neohrozila analytický výstup.

Existující aplikace v zadavatelské společnosti se starají o automatizaci získávání webových dat, jejich katalogizaci, analytickou činnost, popřípadě o vzdálené řízení těchto činností. V rozsahu dat, se kterými firma pracuje, je velice náročná jakákoli editace, popřípadě transformace dat a jejich verifikování. Z toho vyplývá malá výkonnost celého systému, jelikož výkon je omezován množstvím vstupních dat.

Vizí této práce je vytvoření aplikace, jež zrychlí proces editace velkého množství nesourodých a chaotických dat. Toho by mělo být docíleno implementací excel-like řešení, popřípadě exportu a importu zpracovaných dat právě do aplikace Excel. Ke zrychlení a především zjednodušení procesu doposavad ručně tvořených a editovaných dat by také měly pomoci transformační funkce, které umožní ze základních dat získat obtížně zjistitelné informace. K tomu by měly sloužit nadefinované parametrické výpočetní funkce uložené v databázi, které vzniknou jako součást aplikace. Zajištění aktuálnosti by měla dopomoci adaptační část aplikace, jež by měla umožnit verifikované posouvání datumů a dalších dat s nimi souvisejících.

1 Seznámení s ekosystémem

Veškerá data, která jsou použita v této práci, jsou fiktivní nebo jsou to data vlastní, popřípadě pouze ilustrativní. Žádný z uvedených příkladů není konkrétní a veškerá podobnost je náhodná. Vzhledem k bezpečnostní politice firmy a know-how užitého při vývoji aplikace Edittool jsou z práce vědomě vyjmuty některé konkrétní informace, které jsou však v dostatečné míře popsány nebo nahrazeny ilustrativními příklady tak, aby nedošlo k poškození žádné ze zúčastněných stran.

S ohledem na multinárodnost firmy, počínaje celým týmem a konče zpracovávanými daty, je mnohdy názvosloví databázových tabulek i tříd a metod, popřípadě proměnných, shlukem českých, anglických a německých názvů. Vzhledem k zaběhlosti těchto názvů si dovoluji ponechat je v těchto podivných formách. Ze stejného důvodu budou v následujícím textu užívány názvy a spojení, která mohou odporovat běžnému smyslu užití, veškeré tyto výjimky však budou dostatečně popsány a definovány.

Vzhledem ke špatné rozlišitelnosti dat (počítačová data) a dat (časový údaj) v českém jazyce (na rozdíl od angličtiny) je pro větší přehlednost celého dokumentu vědomě užíváno nesprávného tvaru "**datumy**"a ostatních příslušných tvarů. Prosím tímto čtenáře o pochopení a shovívavost při čtení následujících řádků.

1.1 Zadání aplikace

Zadání aplikace není jednotné, ucelené a mnohdy ani zcela jasné. Největší důraz byl kladen na konzistentnost výsledků, rychlost výpočtů a modulárnost v tom ohledu, aby aplikace byla funkční v co nejbližším termínu, přestože s minimem funkcí, které jsou přidávány v průběhu celého procesu.

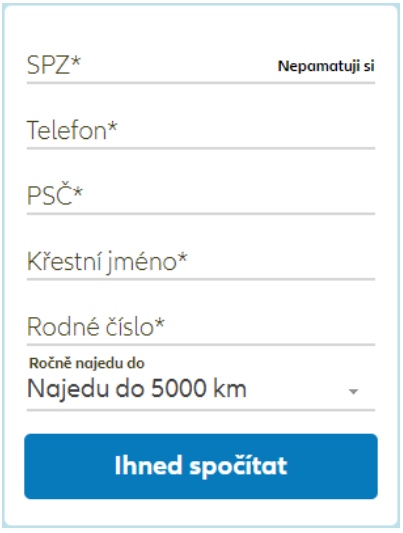
Tato aplikace vzniká od roku 2015 a je psána v jazyce `C#`. Byla částečně vystavěna na již existujících databázových tabulkách a integrována do celého ekosystému. Veškerá řešení jsou výsledkem mé práce. Zadání a integrace celé aplikace pak probíhalo ve spolupráci s hlavním programátorem, který dohlíží na kompatibilitu jednotlivých aplikací a poskytuje znalosti ohledně databáze, a managerem

firmy, jehož představy a know-how byly implementovány v aplikaci. V průběhu let byla aplikace třikrát majoritně přepsána a v budoucnu se nevyhne další obměně. V první vlně se jednalo o jakoukoli automatizaci, kterou dokážeme implementovat. O automatizaci se ve firmě pokoušeli několik předchozích let za pomocí různých nástrojů a programovacích jazyků, pokaždé však neúspěšně. Až do začátku vývoje této aplikace tedy veškerá činnost týkající se vzniku nových dat probíhala ručně s dopomocí nástrojů jako je Microsoft Excel. V tomto režimu byla firma schopna zpracovat jednotky menších datových sad denně.

Stěžejním bodem zadání bylo vytvořit aplikaci, která bude umět data typu A z kolekce subjektu Y převést na data typu A pro kolekci subjektu Z.

Budeme-li se bavit například o pojištění automobilů, představme si tyto kolekce jako vstupní data webových formulářů A a B pojišťoven Y a Z. Veškerá data jsou pak uložena v databázi v tabulce D. Tato databázová tabulka sdružuje všechna vstupní data rozřazená pomocí vazebních tabulek a cizích klíčů. Každý subjekt pak pro vstupní data otázky O_1 používá různé názvy či označení a také její vstupní data se mohou lišit, jak je vidět na obrázku 1.1, kde vidíme úvodní stránky dvou pojišťoven s prvními stránkami jejich formulářů. Databázová entita pro nalezení či vyplnění SPZ bude mít zajisté jiné koordináty, jež se přinejmenším budou lišit textem otázek, stránkou, na které se vyskytují, hlavičkou či typem daného prvku. Právě tyto databázové entity je potřeba navázat takovým způsobem, který bude korespondovat s již vytvořenými a užívanými tabulkami, a umožní tak převedení dat subjektu Y na subjekt Z.

(a) Webový formulář A



(b) Webový formulář B

Obrázek 1.1: Porovnání webových formulářů

Subjekty se dělí na hlavní subjekt, tzv. **master**, a subjekty generované, tzv. **klony**. *Master* pak vzniká použitím formuláře hlavního zdroje, který je rozšířen o další informace, které aplikace dopočítává tak, aby převod mezi subjektem *masteru* a jednotlivými klony byl co nejrychlejší a nepoužíval rozsáhlé a složité výpočty. Vzhledem k počtu klonů, které se pohybují v řádu desítek, je toto žádoucí.

Cílem této aplikace není mapování jednotlivých formulářů, pouze užití jejich dat. Je proto důležité používat v co největší míře data z databáze a vyhnout se užití napevno zakódovaných (hard-coded) specifikací. Valná většina kódu proto musí být psaná parametricky, tak, aby aplikace dokázala reagovat na změny provedené v databázi, nebo v ideálním případě, aby takové informace vůbec nepotřebovala.

1.2 Vstupní data

Vstupní data jsou směsicí dat, jejichž přesný formát ani struktura nejsou definovány. Veškeré zařazení dat probíhá za pomoci ostatních interních nástrojů firmy. Vstupní data jsou částečně unifikována pomocí níže popsaných databázových tabulek (viz 1.2), nad jejich vstupním formátem a strukturou dat však nemáme žádnou kontrolu. Z toho vyplývá rozmanitost a do jisté míry chaotičnost některých níže popsaných postupů, vše ve snaze docílit co nejpřesnějších výsledků v globálním měřítku.

Samotná data jsou strukturována do několika tabulek, z nichž nejdůležitějšími jsou tabulky **dialog_option**, **dialog_question_group** a jejich vazební tabulka **dialog_question_group_ma_dialog_option**, kde se nacházejí samotná vstupní data, a následně tabulky **dialog**, **task** a **vstupni_zdroj_ma_dialog_default**, které sdružují kolekce, zde nazývané *dialogy*, jednotlivých *dialog_option* pro daný formulář, potažmo dané konkrétní pojištění.

1.2.1 Zdroje, portály a pojišťovny

Subjekty, nebo-li zdroje, portály či pojišťovny, mají v celém ekosystému významnou roli. Ke každému subjektu, který je zpracováván, je přistupováno právě dle jeho zařazení. Pro zdroje a portály je žádoucí jiné chování nežli pro pojišťovny. Proto i každý z těchto typů subjektů má svou vlastní tabulku. Pro naší aplikaci však postačí jednotné pojmenování, a to právě **subjekty**.

1.2.2 Databázová tabulka dialog_option

Tabulka **dialog_option** na obrázku 1.2 uchovává data jednotlivých otázek jednotlivých formulářů. Toto se dělí na dvě části, první jsou kolekce dat, které se vyznačují nastavením parametru **used** na **0** a zároveň totožnými hodnotami ve sloupečcích **id_dialog_option** a **id_dialog_option_base**. *Id_dialog_option_base* je vazbou na tuto samou tabulku, v tomto případě na tyto stejné entity, které jsou tzv. **hlavním dialogem** daného zdroje. Jsou to data, ve kterých nalezneme pouze to, co vidíme při načtení webového formuláře, bez toho, aniž by do nich bylo zasaženo člověkem, tudíž zde nejsou žádné vstupní hodnoty. Tyto **dialogy** jsou alfou omegou a obsahují všechny možnosti, které daný webový formulář skýtá.

Column	Type	Comment
page	int(11) *NULL*	
order	int(11) *NULL*	
suborder	int(11) *NULL*	
group_text	text *NULL*	
group_text_custom	text *NULL*	
page_text	text *NULL*	
label	text *NULL*	
radio_text	text *NULL*	
used	int(11) *NULL*	
value	text *NULL*	
radio_text_default	text *NULL*	
used_default	int(11) *NULL*	
value_default	text *NULL*	
isHidden	int(11) *NULL*	
id_dialog	int(10) unsigned	
id_dialog_option	bigint(20) *Auto Increment*	
id_dialog_option_base	bigint(20) *NULL*	
name	text *NULL*	
type	text *NULL*	
path	text *NULL*	
info_text	text *NULL*	
info_additional	text *NULL*	
last_update	datetime *NULL*	
created	datetime *NULL*	

Indexes

PRIMARY	*id_dialog_option*
INDEX	*id_dialog*
INDEX	*id_dialog_option_base*

Alter indexes

Foreign keys

Source	Target	ON DELETE	ON UPDATE	
id_dialog	dialog(*id_dialog*)	CASCADE	CASCADE	Alter
id_dialog_option_base	dialog_option(*id_dialog_option*)	SET NULL	CASCADE	Alter

Obrázek 1.2: tabulka *dialog_option*

Druhou částí funkce této tabulky je uchovávání již konkrétních dialogů. Každá entita je pak pomocí *id_dialog_option_base* svázána se svou mateřskou entitou. Je v plánu přepracovat tyto vazby tak, aby konkrétní entity *dialogů* měly svou vlastní

tabulku, která bude zahrnovat pouze vazbu na mateřskou entitu, svou vlastní reálnou hodnotu, *id_dialog*, popřípadě několik dalších parametrů, které jsou potřeba pro chod celého ekosystému. Bohužel v čase vytváření aplikace nebylo možné provést tento přepis, tudíž je celá konstrukce poněkud kostrbatá.

Vazby mezi entitami a jejich mateřskými entitami jsou jakýmsi prvním krokem k přepsání této databázové části a většina dat potřebných v aplikaci je pomocí vazby přebírána z mateřské entity tak, jak by tomu mělo být po přepsání.

1.2.3 Grupování

Tabulky obsahující informace o *groupách* pro nás nejsou až tak zajímavé, podstatné však je pochopit smysl grupování. **Groupou** nazvěme jeden webový prvek se všemi jeho možnostmi. **Textový** prvek obsahuje právě jednu *dialog_option*. **Radia** obsahují nejčastěji dvě až tři možnosti. **Select-one** obsahuje jednu a více možností. Další prvky, jako například **number**, **tel**, **info** a **button**, jsou pro nás zajímavé pouze v tom ohledu, že jsou považovány za textové. Více se dočteme v kapitole 1.2.5.

Vazby jednotlivých *dialog_option* na jejich *groupy* nalezneme ve vazební tabulce *dialog_question_group_ma_dialog_option*.

Každý jeden *dialog* může obsahovat pouze jednu aktivní (used = 1) *dialog_option* v jedné *groupě*. S variantou multi-select jsme se do současné doby nesetkali, proto nachází-li se v jedné *groupě* více *used* prvků, některý z mechanismů v průběhu celého procesu nepracoval správně.

1.2.4 Databázová tabulka task

Tabulka **task** obsahuje veškeré nastavení potřebné k běhu dalších aplikací v ekosystému. Smyslem *tasků* je udržet stálou informaci o kolekci *dialog_option* pomocí *dialogu*, který tabulka *task* obsahuje jako cizí klíč. *Dialogy* se v čase mohou měnit společně s jejich kolekcemi, avšak *task* konkrétního nastavení a sdruženého identifikátoru zůstává (viz 2.1.2).

1.2.5 HTML input

Chování každé *groupy* je ovlivněno jejím typem. Tento typ je přímo odvozen od typů HTML inputů. K uchování těchto pravidel slouží tabulka **a_variable_types** na obrázku 1.3. Důležitými sloupečky jsou sloupečky **type**, **is_handled_as_text**, **type_of_return**, popřípadě **convert_ja_nein**. Hodnota sloupečku *type* určuje typ

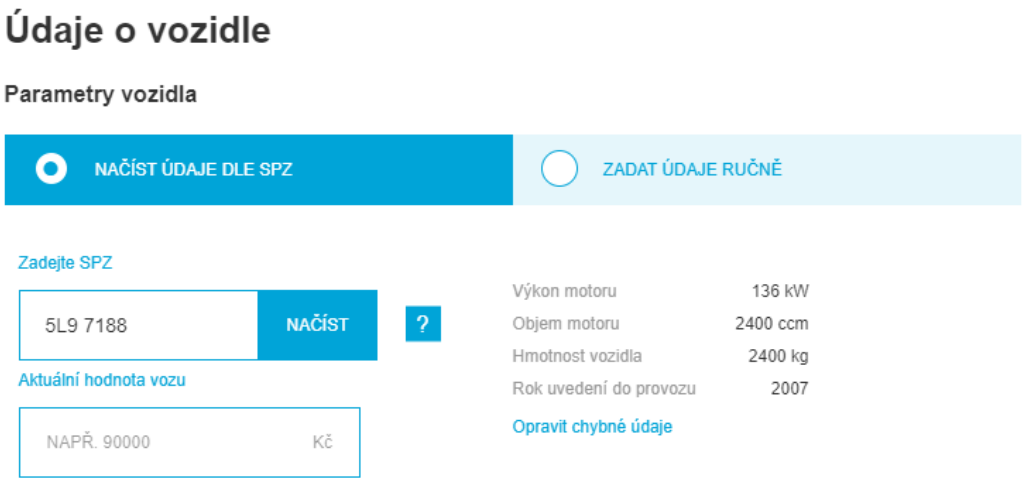
prvku (*groupy* a *dialog_option*), hodnota *is_handled_as_text* určuje, je-li prvek považován za textový, a hotnota sloupečku *type_of_return* určuje, která hodnota se má při zobrazení dané *dialog_option* ukázat. Jedná se o hodnoty z tabulky *dialog_option*, a to buď **radio_text** nebo **value**.

Column	Type	Comment
id_a_variable_test_inputes	int(11) *Auto Increment*	
type	text	
is_handled_as_text	tinyint(4) [**0**]	
type_of_return	text	
not_important	tinyint(4) [**0**]	
ignore_as_new_option	tinyint(4) [**0**]	
set_as_not_visible_tonda	tinyint(4) [**0**]	
convert_ja_nein	tinyint(4) [**0**]	

Obrázek 1.3: tabulka *a_variable_types*

1.2.6 Skutečné hodnoty

Na obrázku 1.4 vidíme vyplněné první dva prvky formuláře. Prvním je zvolení typu zadávání vozidla pomocí SPZ. Druhým prvkem je zadaná SPZ.



Obrázek 1.4: Vyplněný formulář

Na obrázku 1.5 vidíme, jak by zjednodušeně vypadal *master dialog* pro tento portál. Na obrázku 1.6 lze vidět, jak by vypadal reálný *dialog* tohoto konkrétního zadaného formuláře.

Modify	page	order	suborder	group_text	page_text	label	radio_text	used	value	id_dialog	id_dialog_option	id_dialog_option_base	type
edit	1	0	0	Údaje o vozidle	vstupni-udaje	Parametry vozidla	NAČÍST ÚDAJE DLE SPZ	0		123456	987654	987654	select-one
edit	1	0	1	Údaje o vozidle	vstupni-udaje	Parametry vozidla	ZADAT ÚDAJE RUČNĚ	0		123456	987655	987655	select-one
edit	1	1	0	Údaje o vozidle	vstupni-udaje	Zadejte SPZ		0		123456	987656	987656	text

Obrázek 1.5: *Master dialog* v DB

Modify	page	order	suborder	group_text	page_text	label	radio_text	used	value	id_dialog	id_dialog_option	id_dialog_option_base	type
edit	1	0	0	Údaje o vozidle	vstupni-udaje	Parametry vozidla	NAČÍST ÚDAJE DLE SPZ	1		176802	3006321313	987654	select-one
edit	1	0	1	Údaje o vozidle	vstupni-udaje	Parametry vozidla	ZADAT ÚDAJE RUČNĚ	0		176802	3006321314	987655	select-one
edit	1	1	0	Údaje o vozidle	vstupni-udaje	Zadejte SPZ		1	5L9 7189	176802	3006321315	987656	text

Obrázek 1.6: Konkrétní *dialog* v DB

1.3 Softwarové řešení a možnosti

Vzhledem k začleňování aplikace do již běžícího procesu a zaběhlých řešení firmy bylo žádoucí užití stejných softwarových řešení, která již byla používána v ostatních aplikacích firmy.

1.3.1 Databázová řešení

Firma ke katalogizaci svých dat užívá relační databáze **MySQL** z balíčku XAMPP [1]. Vzhledem k licenční politice tohoto řešení, jejímu výkonu a již implementovaným řešením bylo v této práci navázáno na totéž.

Nejčastěji užívaným řešením z pohledu přístupů k databázi je využití čistých SQL dotazů a insertů (MySQL Connector[2]) ve spojení s Entity Frameworkem verze 6 (dále EF6, viz níže). Toto spojení se ukázalo býti vhodným z hlediska rychlosti čistého SQL (například při parametrizovaných SQL insertech či načítání velkého množství dat) a jednoduché implementace a snadnosti ladění EF6, včetně objektově relačního zobrazení databázových entit pro pohodlnější zpracování veškerých dat.

Okrajovým řešením je Telerik Open Acces (viz níže), který byl využit pro vývoj pouze jedné aplikace a který díky svým licenčním podmínkám nepřipadá nadále v úvahu.

1. **MySQL**

   "*MySQL je relační databázový systém s otevřeným zdrojovým kódem založený na SQL. Byl navržen a optimalizován pro webové aplikace a může běžet na libovolné platformě. Jak se v souvislosti s rozvojem internetu objevily nové a odlišné požadavky na databázi, MySQL se stala oblíbenou platformou pro webové vývojáře a webové aplikace. Protože je navržena tak, aby zpracovávala miliony dotazů a tisíce transakcí, je databáze MySQL oblíbenou volbou pro elektronické obchody, které potřebují zajišťovat hromadné peněžní transakce. Primárním rysem MySQL je flexibilita na vyžádání.*" [1]

[1] Dostupné z: https://www.apachefriends.org/index.html
[2] Dostupné z: https://www.mysql.com/products/connector/

2. **Enity Framework 6**

   "*Enity Framework 6 (EF6) je vyzkoušený a testovaný objektově-relační Mapovač (O/RM) pro .NET s mnoha roky vývoje a stabilizací funkcí. EF6 v/RM snižuje nesoulad mezi relačními a objektově orientovanými světy a umožňuje vývojářům psát aplikace, které pracují s daty uloženými v relačních databázích pomocí objektů .NET se silnými typy, které představují doménu aplikace, a eliminují nutnost velké části kódu nutné k přístupu k datům, které obvykle potřebují zapisovat.*" [2]

3. **Telerik Data Access**

   "*Telerik Open Access, respektive Telerik Data Access je nástroj, který podporuje vývoj softwarově orientovaných softwarových aplikací. Nástroj Telerik Data Access je zaměřen na řešení nesouladu objektově-relačních impedancí.*" [3]

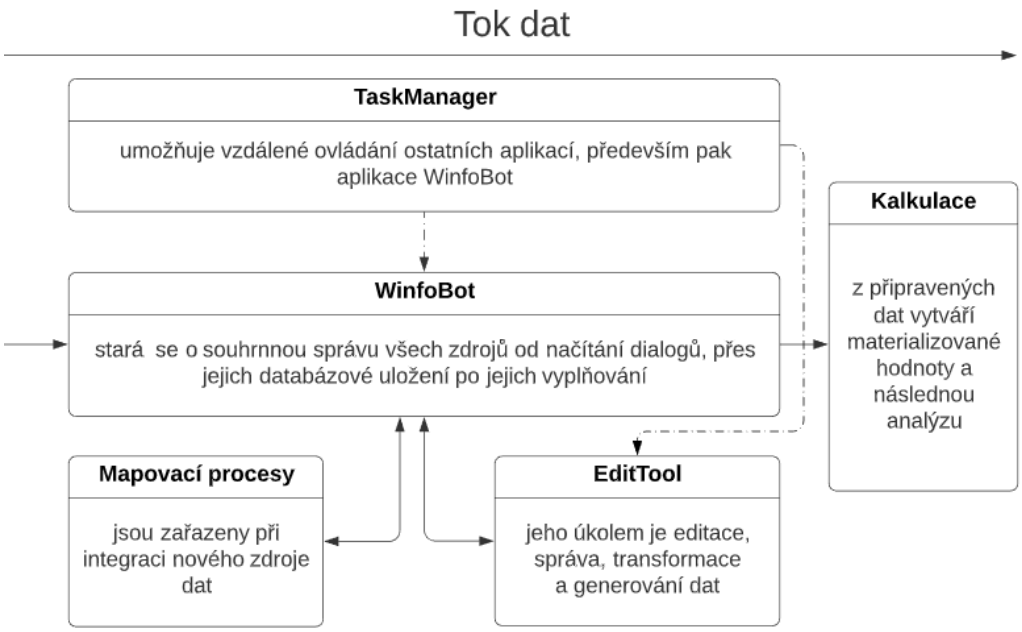
1.3.2 Načítání a ukládání dat - Microsoft Excel

Valná část dat zpracovávaná aplikací je exportována do Excelu, přesněji do formátu .xlsx. .xlsx je formát dat společnosti Microsoft využívaný v Microsoft Excel 2007 a novějších. Byl zvolen pro jeho aktuálnost. Je běžným výstupní formátem pro data z Microsoft Excel. Pro import i export dat je využíváno doplňkového balíčku EPPlus, jenž umožňuje rychlou a efektivní práci s daty a je rovněž zaběhlým řešením ve firmě. Tento doplněk je vhodný především pro importování dat, kde je lepším řešením než oficiální knihovna Microsoft Interop Excel. Ta se ovšem ukázala jako lepší možnost pro exportování dat. Ani jedno řešení není optimální, jejich spojením však vniká uspokojivé řešení pro import a export dat. Tento postup je ve firmě běžným řešením a alternativy k těmto řešením se hledají jen velmi těžko.

1.3.3 Interakce s ostatními aplikace

Vzhledem k provázanosti celého ekosystému bylo nutné částečné či úplné propojení aplikace Edittool a ostatních aplikací. Ačkoli se nejedná o hlavní problém této práce, je dobré zmínit, jak toho bylo docíleno. Aplikace Edittool implementuje několik dalších projektů. První projektem je projekt zajišťující komunikaci s tzv. **TaskManagerem**. To umožňuje spouštět aplikaci Edittool ve stavu **klienta**, což umožňuje vzdálené využívání hlavní funkcionality aplikace. Implementace proběhla pro hlavní část, tedy pro generování dat. Dalším implementovaným projektem je projekt ověřující aktuálnost databázových dat pro automobily. Tento projekt v sobě

zahrnuje strojové ovládání prohlížeče Firefox pomocí automatizačního softwaru **Selenium**. Bylo tedy nutné seznámit se a pochopit všechny výše zmíněné softwary tak, aby jejich implementace proběhla správně. Vše dle dokumnecace k aplikaci WinfoBot [4]. Pro pochopení všech souvislostí aplikací bylo mimo jiné zapotřebí seznámit se s data miningem [5]. Pro lepší pochopení propojení aplikací a jejich činností poslouží obrázek 1.7.



Obrázek 1.7: Datový tok veškerých dat aplikacemi

2 Funkcionalita aplikace

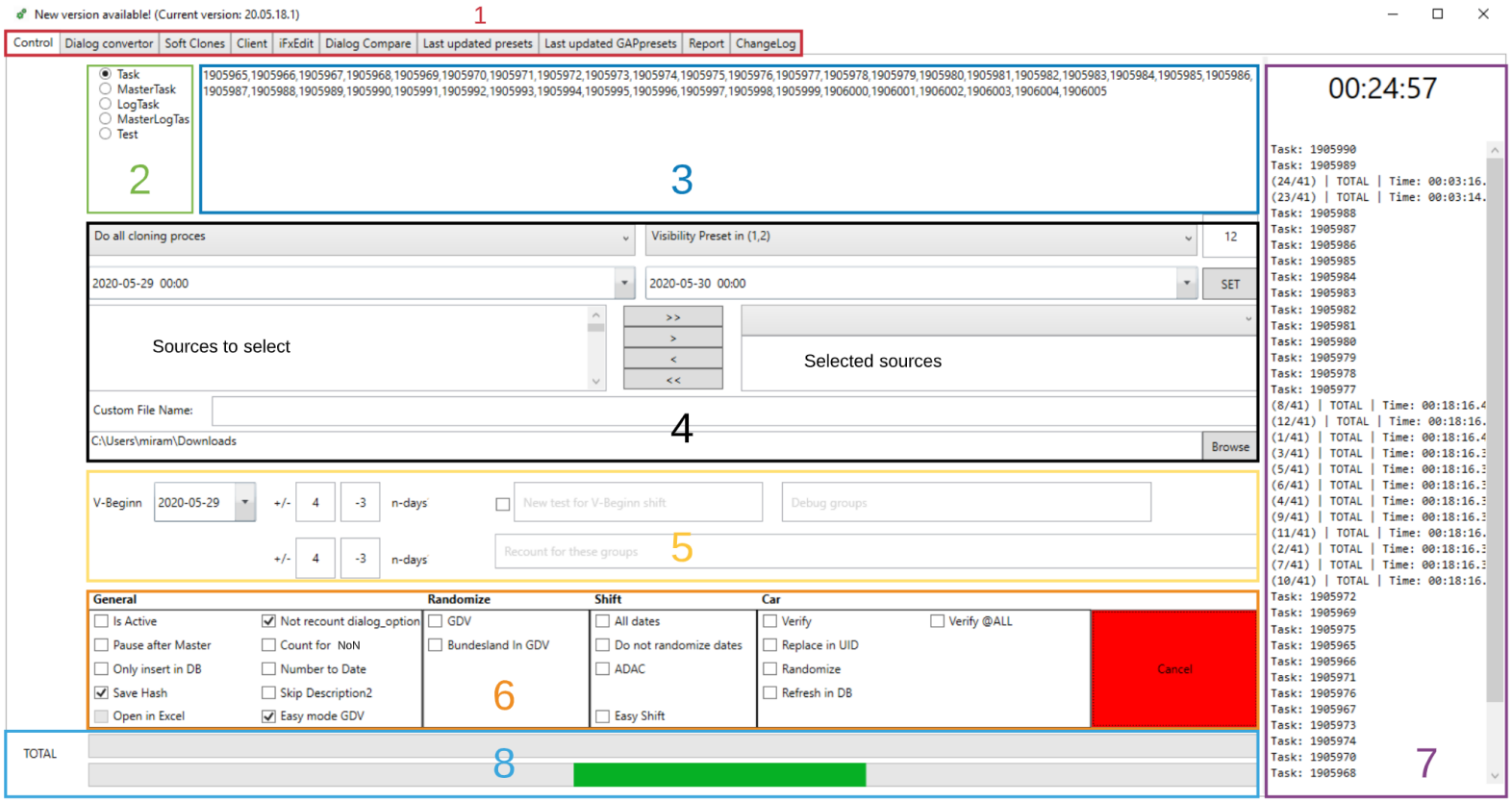
2.1 GUI

Prvním hlavním úkolem, který musela aplikace splňovat, bylo importování a exportování dat, potažmo zobrazování dat. Toto bylo řešeno pomocí grafického rozhraní (na obrázku 2.1), které umožní importovat a exportovat zvolená data do excelu a zároveň data zobrazí v excel-like komponentě.

V horní části aplikace v červeně označené sekci **1** se nacházejí jednotlivé záložky aplikace. Aktuálně zobrazená je záložka **Control**. Funkce některých dalších záložek bude popsána v jednotlivých kapitolách dále. Na aktuální záložce *Control* se nachází hlavní nastavení aplikace. V zeleně označené sekci **2** se nachází skupina *radio buttonů*, které rozhodují o vyhledávacím algoritmu exportovaných dat. V tomto případě je aplikace nastavena nejběžnějším způsobem, tedy zadáváním *tasků*. Ty samotné se zadávají to textového okna v modré sekci **3**, kde vidíme kolekci právě zadaných *tasků*. Jejich separátorem je '**,**'. Níže, v **černé** sekci **4**, se nachází nastavení módu exportu, v tomto případě **Do all clonning proces**, čili generování *masterů* a generování *klonů* z *masteru*. Nejběžnější volbou módu je pak **Base Dialogs**, který dohledá aktuální *dialog* pro daný *task*. Datumy níže určují datumové rozpětí vzniku *tasků*, které jsou vyhledávány v databázi. **Sources to select** a **Selected sources** jsou listy, pomocí kterých se řídí samotné generování, popřípadě jsou to filtry určující, které subjekty se mají exportovat. V této sekci se také nachází pole pro přejmenování exportovaného souboru a pole se systémovou adresou. To slouží k nastavení systémové složky pro export, nebo pro importování souboru pomocí drag&drop. Datum a čísla níže ve žluté sekci **5** jsou nastavením **DateShiftu**, o jehož fungování se dočteme v kapitole 2.3.1. V pravé části této sekce se nachází několik textových polí využívaných pro debuggování. Ve spodní části v oranžové sekci **6** se nachází nastavení či donastavení jednotlivých módů aplikace. Toto nastavení je částečně určeno výběrem módu aplikace a může být donastaveno uživatelem. Nastavení je rozděleno do jednotlivých podsekcí tak, aby se zpřehlednilo nastavování

a vyjasnil význam jednotlivých možností. Červené tlačítko **Cancel** slouží v běžícím stavu k přerušení operací a zastavení aplikace. V nebežícím stavu je pak spouštěčem veškerých procesů. Pravá strana aplikace (fialová sekce **7**) slouží jako informační panel, kde jsou zobrazovány všechny důležité zprávy, kontrolní hlášky či chyby a také veškerý pokrok v daném procesu. Pro rychlý přehled slouží také dva ukazatele průběhu umístěné ve spodní části aplikace ve světle modré sekci **8**.

Vzhled celé aplikace není nikterak oslnivý, je však funkční a dostačující. Aplikace není grafickým produktem. Jedná se o interní nástroj, proto vzhledu není přikládán téměř žádný důraz. Barevné sekce na obrázku 2.1 byly doplněny pro účel lepší popisnosti uživatelského rozhraní a v běžném provozu se nevyskytují.



Obrázek 2.1: GUI aplikace EditTool

2.1.1 Import a Export

Import a export dat pro samotné *tasky*, respektive jejich *dialogy*, byl implementován pomocí doplňkového balíčku **EPPlus** a **interop excel**. Vzhled ilustrativního exportovaného dokumentu si můžete prohlédnout na obrázku 2.2.

	A	B	E	I	J	K	L	M	N	O	P	Q
1	HEAD	-100			Id Navigacni Test							
2	HEAD	-200			Id Navigacni Base	0	1	2	3	4	5	6
3	HEAD	-300			Id Navigacni	0	1	2	3	4	5	6
4	HEAD	-400			Tag	#PRESETS	#PRESETS	#PRESETS	#PRESETS	#PRESETS	#PRESETS	#PRESETS
5	HEAD	-500			Test	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1	#1234567# Nájezd_KM_ preset #v200526-1
6	HEAD	-600			UID	#Najezd= 0.000 km	#Najezd= 1.000 km	#Najezd= 2.000 km	#Najezd= 3.000 km	#Najezd= 4.000 km	#Najezd= 5.000 km	#Najezd= 6.000 km
7	HEAD	-700			Description							
8	HEAD	-701			Description2							
9	HEAD	-800			Report	#PRESETS #20	#PRESETS #20	#PRESETS #20	#PRESETS #20	#PRESETS #20	#PRESETS #20	#PRESETS #20
10												
18	MASTER	-13	1	Dialog Info	iFx_Comment_UID							
1062	MASTER	2411	1	Najezd	Roční nájezd(.000 km)	0	1	2	3	4	5	6
2932	subj. A	397200	1	Najezd	Najezd (Km)	do 1.000 km	do 1.000 km	do 2.000 km	do 3.000 km	do 4.000 km	do 5.000 km	do 6.000 km
3274	subj. B	749317	1	Najezd	Najezd (Km)	do 5.000 km	do 5.000 km	do 5.000 km	do 5.000 km	do 5.000 km	do 5.000 km	do 10.000 km

Obrázek 2.2: Exportovaná data

Sloupce[1] **B** až **J** obsahují informacemi o dané groupě. V hlavičce (**HEAD** ve sloupci **A**, řádky **1** až **9**) se pak nacházejí informace o jednotlivých složených identifikátorech. Sloupce **K** a další obsahují zvolené *dialog_option*, respektive jejich *radio_texty* či hodnoty (*value*). Jednotlivé subjekty, respektive jejich *dialogy*, se nacházejí vždy pod jejich složeným identifikátorem. To umožňuje přehledné zobrazení svázaných *dialogů*, stejně jako zobrazení velkého množství různých složených identifikátorů.

Hlavní výhodou využití excelu je:

- **Snadné a přímé zálohování dat**

  Vzhledem k četnosti změn, které probíhají v databázi na úrovni dat a jejich informací, je toto zálohování ideálním řešením. Data uložená v excelu obsahují krom samotných hodnot jednotlivých *dialogů* zároveň informaci o tom, jak vypadal mateřský *dialog* daného subjektu v době exportu. Samotné hodnoty pak umožňují kontrolu funkčních *group* a správnosti jejich výpočtu vzhledem k novějším funkcím. Díky verzování aplikace, které je ve formátu **yy.MM.dd.*publikovaná_verze_toho_dne***, pak lze snadno dohledat příslušnou verzi a změny v kódu.

[1] Skryté sloupce pro nás neobsahují důležité informace a pro větší přehlednost byly vynechány ze zobrazení. Rovněž řádky zobrazení byly vybrány tak, aby obsahovali zajímavá a popisná data.

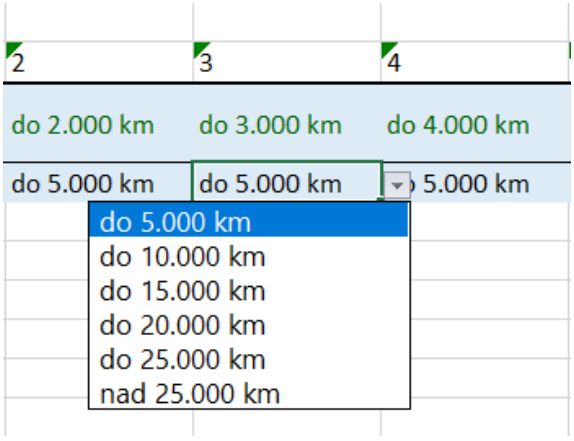
- **Možnost rozsáhlé editace bez vytváření složitého excel-like řešení**

  Aplikace samotná obsahuje excel-like komponentu, která se zobrazí vždy po zpracování dané datové sady. Data zde nejsou zobrazena ve sloupci, kterému náleží sdružený identifikátor, pod sebou, nýbrž jsou rozdělena do vlastních záložek. Tato možnost slouží pro rychlou kontrolu, obsahuje několik kontextových voleb, ale v průběhu používání aplikace byla tato možnost odsunuta do pozadí a upřednostněna editace v aplikaci Excel.

- **Využití již zaběhlého softwaru, jehož funkce jsou rozsáhlé**

  Jak již bylo popsáno výše, aplikace Excel je silným nástrojem. Umožňuje například jednoduché a rychlé filtrování hodnot, uzamčení (freeze) sloupců i řádků (využito u hlavičky a informacích o *groupách*), vytváření drop-down buněk a mnohé další. Ve firmě je též využíváno vlastních doplňků do excelu a analyzačních funkcí excelu.

Velkou výhodou je výše zmíněná možnost vytváření dropdown[2] buněk, které vidíme na obrázku 2.3. To je řešeno pomocí skrytého sheetu, kde jsou ve sloupcích vedle sebe umístěna čísla group a pod nimi se nachází jejich hodnoty. Pomocí **Data validation** jsou pak tyto hodnoty umístěny do vlastních buněk jednotlivých group. Toto řešení zabraňuje copy-paste chybám a překlepům při editaci dialogu.



Obrázek 2.3: Excel dropdown

Každé nastavení exportu má svůj specifický název, který je přidán i do názvu vytvořeného souboru. To v první řadě zvyšuje přehled v exportovaných souborech a v druhé řadě se jím řídí některé importovací funkce, které tak jednoduše rozeznají, o jaký druh importu se jedná.

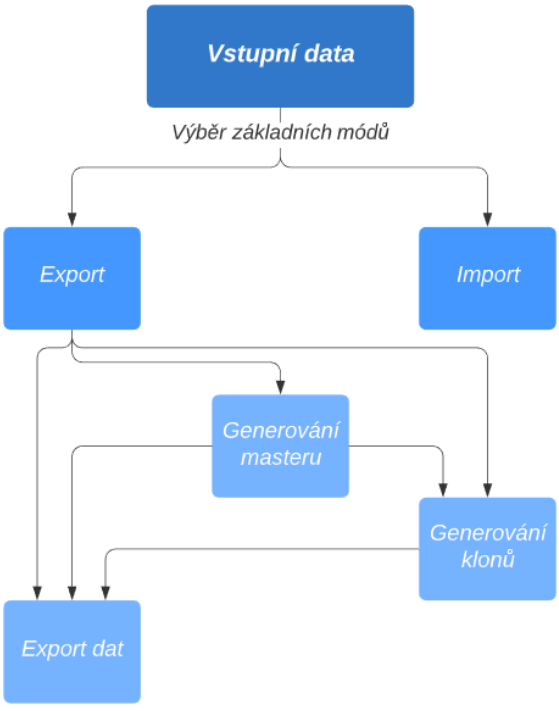
[2]https://support.office.com/en-us/article/create-a-drop-down-list-7693307a-59ef-400a-b769-c5402dce407b

2.1.2 Složený identifikátor

Složený identifikátor je spojením několika hodnot z tabulky *task*, které jsou vazbou mezi jednotlivými subjekty. Při generování nových *tasků* (*klonů*) je díky této vazbě zachována souvislost *tasků*, potažmo *dialogů*. Každý subjekt může obsahovat právě jeden *task* daného složeného identifikátoru. Tento identifikátor dává zasvěcenému jasný přehled o tom, jaký typ dat daný *task*, potažmo *dialog*, obsahuje. Jak vidíme na obrázku 2.2, skládá se ze tří číselných hodnot (**Id navigacni test**, **Id navigacni base** a **Id navigacni**) a dále z pěti textových hodnot (**Tag**. **Test**, **UID**, **Description** a **report**).

2.1.3 Základní tok dat v aplikaci

Aplikace dělí tok dat na dvě základní větve, jak můžeme vidět na obrázku 2.4, na **Import** a **Export**. Při exportu dat jsou dle zvoleného módu využity metody **Generování masteru**, potažmo **masterů**, a **Generování klonů**. Tento výběr částečně závisí i na vstupních datech. Samotný export dat obsahuje další pokročilé funkce pro transformaci dat, jak bude popsáno níže.



Obrázek 2.4: Tok dat aplikací Edittool

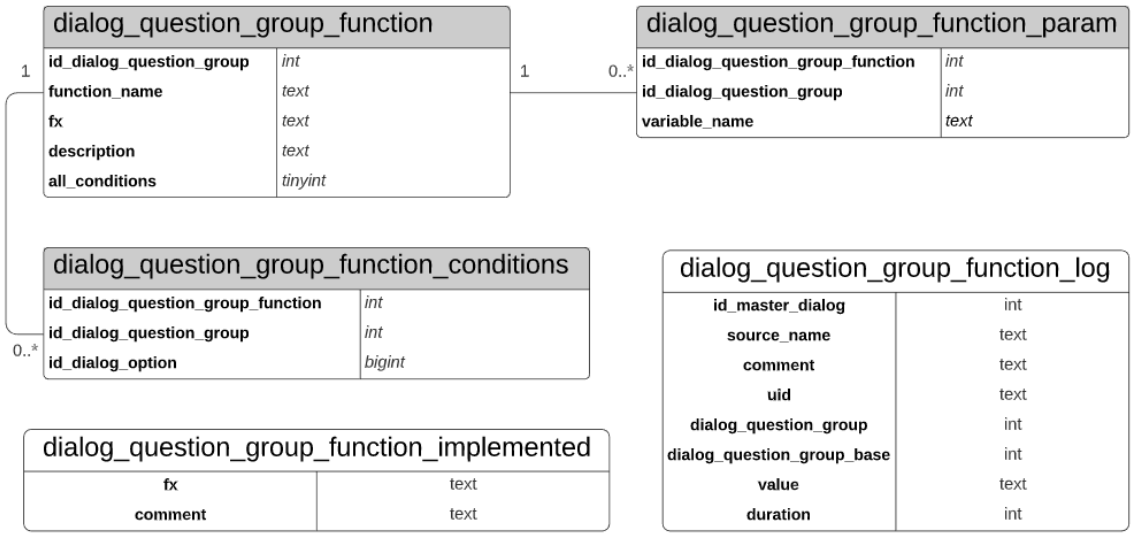
2.2 Výpočetní funkce

Dosavadní informace o datech nám dávají dostatečný přehled o tom, s čím aplikace pracuje. Aplikace k výpočtům používá funkce, které by se daly nazvat jako excel-like funkce. Tyto funkce jsou společně s jejich parametry a podmínkami uloženy v několika databázových tabulkách. Jejich smyslem je určit, jak se vypočítá hodnota konkrétní groupy. Ta se zpravidla vypočítá z jedné a více group, ať již reálných nebo vypočítaných pomocí jiné funkce (funkční *groupa*).

Rychlý přehled o funkcích si můžeme udělat z obrázku 2.5, kde vidíme databázové tabulky týkající se funkcí. Jak vidíme, funkce (**dialog_question_group_function**) může mít **n** podmínek a **m** parametrů. Samotnou funkci pak tvoří *groupa*, jejíž hodnotu funkce vypočítává, popisné pojmenování funkce, identifikační pojmenování (např. **SUMA** v excelu), popis (ten může obsahovat doplňující informace pro samotný výpočet funkční *groupy*. Vyhodnocení popisu probíhá pomocí regulárních výrazů [6]) a typ vyhodnocování podmínek.

Podmínky (**dialog_question_group_function_conditions**) jsou pak navázány na danou funkci a obsahují informaci o *dialog_option*, která tvoří onu podmínku, a o *dialog_question_group*, ze které *dialog_option* pochází.

Parametry (**dialog_question_group_function_param**) jsou opět navázány na danou funkci a obsahují informaci o *dialog_question_group*, která poskytuje zdrojová data pro výpočet, a **variable_name**, což je pojmenování zdrojové *groupy* pro lepší přehlednost.



Obrázek 2.5: Zjednodušený UML diagram funkcí a jejich tabulek

Dvě doplňující tabulky **dialog_question_group_function_implemented** a **dialog_question_group_function_log** jsou informačními tabulkami. První z nich (**dialog_question_group_function_implemented**) pouze uchovává informace o již implementovaných funkcích, zatímco druhá (**dialog_question_group_function_log**) slouží k logování výpočtu jednotlivých funkcí. Tato tabulka poskytuje cenné informace o trvání jednotlivých výpočtů a lze zde porovnávat i vypočtené hodnoty.

V aplikaci je momentálně implementováno více než 50 funkcí s více než 80 rozdílnými voláními a databáze obsahuje více než 3700 databázových záznamů jednotlivých funkcí.

2.2.1 Podmínky

Výše zmíněný typ vyhodnocování podmínek platí vždy pro celou kolekci podmínek navázaných na danou funkci. V následujícím bloku kódu je patrné vyhodnocování podmínek. Celkem se jedná o 4 případy (popsáno níže v kódu) rozšířené o defaultní možnosti.

```
switch (all_conditions)
    {
        case 0: // alespon jedna splnena
        case 10:
            condition = conditions.Intersect(this.taskInfo.
                DialogOptionsConditions).Any()
            || !conditions.Any();
            break;
        case 1: // vsechny splnene
        case 11:
            condition = !conditions.Except(this.taskInfo.DialogOptionsConditions
                ).Any();
            break;
        case 2: // neobsahuje vsechny tyto dialog_option
        case 12:
            condition = conditions.Except(this.taskInfo.DialogOptionsConditions)
                .Any();
            break;
        case 3: // neobsahuje alespon jednu dialog_option
        case 13:
            condition = !conditions.Intersect(this.taskInfo.
                DialogOptionsConditions).Any();
            break;
    }

    if (!condition && all_conditions >= 10) //default
    {
        FunctionsVariables.Add(this.id_dialog_question_group, taskInfo, this.
            DefaulValue());
    }
    return !condition; // is not hit
```

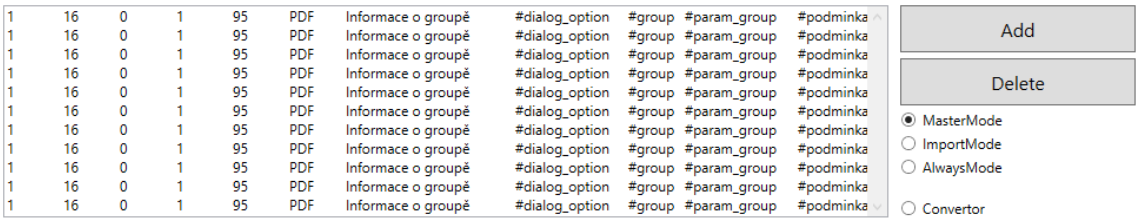
Pro textové *groupy* je defaultní hodnotou text, který se nachází ve sloupečku *value_default* z tabulky *dialog_option*, jenž náleží jediné *dialog_option* z dané *groupy*, jejíž hodnota *used_default* je nastavena na 1.

Pro netextové *groupy* je defaultní hodnotou nikoli text této *dialog_option*, jako v předcházejícím případě, ale celá tato mateřská *dialog_option*.

Díky této hodnotě se zjednodušuje práce s pravidly i samotnými funkcemi, jelikož nevyhodnocení groupy je mnohdy jasným indikátorem toho, že hodnota groupy má být právě defaultní hodnota, potažmo defaultní *dialog_option*. V některých případech je tato možnost zneužita jako jakási podmínka v podmínce, není pak třeba vytvářet speciální podmíněnou funkci.

2.2.2 Správa funkcí

Ze zadání aplikace také vyplývá, že aplikace musí být schopna veškerá data importovat a exportovat do excelu. Nejinak tomu je právě u funkcí, které jsou pomocí tabu **iFxEdit** (na obrázku 2.6) nahrávány ve formátu tsv (**Tab-Separated Values**). Nejedná se o nahrávání načteného souboru, pouze o copy-paste zvolených funkcí přímo z excelu. Jakkoli se toto může zdát jako nepraktické, umožňuje to jednoduché a rychlé nahrání jedné funkce, stejně tak jako nahrání celého souboru. Samotné nahrávání se pak stará o správné zařazení funkcí do tabulek. Posledním krokem nahrávání, které vzniklo jako poslední významná část aplikace, je konverze funkcí do konvertovacích pravidel.



Obrázek 2.6: Ukázka importování funkcí

2.2.3 Konvertovací pravidla

Tato pravidla jsou jakousi zjednodušenou verzí funkcí. Vzhledem k podobnosti některých subjektů se mapování *dialog_option* dá provádět přímou vazbou *dialog_option* 1:1. V takovém případě přicházejí na řadu konvertovací pravidla. V databázové tabulce, jejíž strukturu vidíme na obrázku 2.7, jsou jednotlivé vazby *dialog_option* na *dialog_option*. I tato tabulka umožňuje použití podmínek.

Column	Type	Comment
id_dialog_option_convertor	int(11) unsigned *Auto Increment*	
id_vstupni_zdroj_left	int(10) unsigned	
id_pojistovna_left	int(10) unsigned *NULL*	
id_vstupni_zdroj_right	int(10) unsigned	
id_pojistovna_right	int(10) unsigned *NULL*	
id_dialog_option_base_left	bigint(20)	
id_dialog_option_base_right	bigint(20)	
group	int(11) *NULL*	
all_condition	int(11) [**0**]	
condition	text	use as condition
fx	text	extra function to test
bug_comment	text	use as bug_comment
zmena	datetime	
vytvoren	datetime	

Indexes

PRIMARY	*id_dialog_option_convertor*
INDEX	*id_vstupni_zdroj_left*
INDEX	*id_vstupni_zdroj_right*
INDEX	*id_dialog_option_base_left*
INDEX	*id_dialog_option_base_right*
INDEX	*id_pojistovna_left*
INDEX	*id_pojistovna_right*

Alter indexes

Foreign keys

Source	Target	ON DELETE	ON UPDATE	
id_vstupni_zdroj_left	vstupni_zdroj(*id_vstupni_zdroj*)	CASCADE	RESTRICT	Alter
id_vstupni_zdroj_right	vstupni_zdroj(*id_vstupni_zdroj*)	CASCADE	RESTRICT	Alter
id_dialog_option_base_left	dialog_option(*id_dialog_option*)	CASCADE	RESTRICT	Alter
id_dialog_option_base_right	dialog_option(*id_dialog_option*)	CASCADE	RESTRICT	Alter
id_pojistovna_left	pojistovna(*id_pojistovna*)	RESTRICT	RESTRICT	Alter
id_pojistovna_right	pojistovna(*id_pojistovna*)	RESTRICT	RESTRICT	Alter

Obrázek 2.7: tabulka dialog_option _convertor

Vzhledem k tomu, že konvertovací pravidla vznikala nejprve jako samostatná část, která měla umět převádět data z PDF formátu na hlavní subjekt, je její implementace v kódu oproti funkcím odlišná a jednodušší. Bohužel však neumožňuje přímý výpočet jedné funkce. Tato možnost se však pro vybrané funkce nahradila složitějším kódem konverze a je prováděna pouze pro pravidla, jejichž hodnota sloupečku **fx** není prázdná. Tato pravidla jsou právě výše zmiňovanou konverzí do konvertovacích pravidel. Jedná se o převedení funkce, jež je zadána jako *group-group* vazba, na vazbu *dialog_option-dialog_option*. Tato konverze probíhá dle zadaných pravidel pro jednotlivé typy *group*, rozřazených do tří funkčních bloků.

- use DEFAULT - použije defaultní hodnotu z tabulky *dialog_question_group*
- use USED, use PRESET, *single dialog_option* - nastaví zdrojovou *dialog_option*

jako 0 pro *use PRESET* a *use USED*, pro *single dialog_option* použije její hodnotu

(*id_dialog_option* 0 je placeholder právě pro případy, kdy výstupní *dialog_option* není přímo závislá na zdrojové *dialog_option*. V podstatě se jedná o podmíněné umístění dané výstupní *dialog_option*, které je závislé nikoli na vstupní kolekci, ale na podmínce vztažené k výstupní kolekci)

- use TXT, *multiple dialog_options* - nalezne párovou schodu *value A* = *value B*

Nulová zdrojová *dialog_option* je v procesu konverze zpracovávána jako defaultní hodnota z mateřské *dialog_option*, nebo jako **preset**. O fungování *presetů* se dočteme níže.

2.2.4 Presety

Složitější funkce, respektive hodnoty, které závisejí na větším množství *group*, byly implementovány jako tzv. **presety**. Jejich smyslem je nastavení selektové *groupy* s obvykle větším počtem možností v závislosti na hodnotách obsažený či neobsažených v jiných *groupách*, bez nutnosti vytvářet další specializované funkce. Tyto *presety* jsou vytvářeny v excelu, z kterého jsou následně importovány do databáze. Více o importu a exportu do excelu se dočteme v kapitole 2.1.1.

Na obrázku 2.2 vidíme, jak by mohl vypadat jednoduchý preset mapující nájezd kilometrů *master*u na jednotlivé subjekty. Tento způsob umožňuje rychlou změnu velkého množství dat bez specifikování zbytečně složitých funkcí a bez zásahu do kódu či funkcí.

Na příkladu *presetů* si ukážeme, jak fungují samotné funkce.

- Názvy funkcí jsou ve formátu **iFx_*název_funkce***
- Funkce implementuje *interface* **IMapping** a její *bází* je *abstaktní* třída **AMapping**
- Pokud není splněna podmínka výpočtu (viz kapitola 2.2.1) [*this.IsConditionNotHit*]
- **nebo** pokud již funkce byla volána se stejnými parametry nebo její hodnotu aktuální dialog obsahuje (závisí na nastavení běhu aplikace) [*IsNotInOrNull()*] $\longrightarrow$ výpočet neprobíhá
- následuje samotná logika funkce

- metoda zpravidla končí voláním metody
  FunctionsVariables.Add(this.id_dialog_question_group , this.taskInfo , Object);

```
1  public class iFx_Preset : AMapping, IMapping
2      {
3          public iFx_Preset(FunctionsVariables FunctionsVariables, TaskInfo
               taskInfo, long id_dialog_question_group)
4              : base(FunctionsVariables, taskInfo, id_dialog_question_group, false
                   , true)
5          {
6          }
7
8          public void Calculate()
9          {
10             if (this.IsConditionNotHit || IsNotInOrNull())
11             {
12                 foreach (long item_affected in HelperPreset.FindAllAffectedRows(
                       this.id_dialog_question_group).Where(x => x != this.
                       id_dialog_question_group))
13                 {
14                     new FunctionController().ReturnCalculation(
                           FunctionsVariables, item_affected, taskInfo,
                           id_dialog_question_group);
15                 }
16
17                 //if group is info, leave it
18                 var preset = HelperPreset.FindPresetFromPresetGroup(taskInfo,
                       this.Options, this.id_dialog_question_group);
19                 FunctionsVariables.Add(this.id_dialog_question_group, this.
                       taskInfo, preset);
20             }
21         }
22     }
```

2.2.5 Abstraktní třída AMapping

Tato abstraktní třída sdružuje veškeré potřebné informace pro všechny funkce, respektive pro každou z nich zvlášť. Jejím cílem je unifikovat proces vyhodnocování funkcí tak, aby se kód funkce dobře psal a především byl přehledný pro další úpravy.

```
1  public abstract class AMapping
2      {
3          /// <summary> load groups, if !options.Any() -> load options.
4          protected AMapping(FunctionsVariables FunctionsVariables, TaskInfo
               taskInfo, long id_dialog_question_group, bool isDefault = false,
               bool isPreset = false) ...
5          public bool isPreset { get; set; }
6          public bool isDefault { get; set; }
7          public long id_dialog_question_group { get; set; }
8          public long id_dialog { get; set; }
9          public long[] Params { get; set; } //all param groups
10         public long[] Groups { get; set; } //all real groups from params
11         public long[] Functions { get; set; } //all function groups from params
12         public FunctionsVariables FunctionsVariables;
13         public TaskInfo taskInfo;
14         public List<my_dialog_option> Options ... //recalculate all groups
15         public bool IsConditionNotHit ... //gets isConditionNotHit() value (just
               for simply writing...)
16         private bool isConditionNotHit() ...
17         public bool isConditionNotHit(long group) ... //recalculate all group
               for conditions and return negative condition result
18         public bool IsNotInOrNull(bool DoNotRecountAnyway = false) ... //
               accoring to the setting returns permition to recalculate this fx
19         public void LoadDialogOptions() ... //reload options for dialog
20         public bool LoadGroups() ... //reload and regroup groups for this fx
21         public my_dialog_option GetOption(long group) ... //return option for
               group from taskInfo
22         public my_dialog_option GetOptionForAgeCount(long group) ... //return
               used option from DialogOptions for selected group
23         public DefaultValue DefaulValue() ... //return default value for this fx
24     }
```

2.2.6 Interface IMapping

Jedná se o jednoduchý interface zajišťující stejnou strukturu všech funkcí dle díla Clean C# [7].

```
1  public interface IMapping
2      {
3          void Calculate();
4      }
```

2.2.7 Object FunctionsVariables

Jak bylo popsáno výše a jak je vidět v kódu abstraktní třídy, dalším důležitým prvkem je objekt **FunctionalVariables**. Kromě ostatních informací důležitých k jeho správnému fungování obsahuje především rozsáhlou metodu **Add(long group, TaskInfo taskInfo, object o, bool result = true))**. Ta se stará o vyhledání a uložení správné *dialog_option* do kolekce. Tím je opět unifikováno přidávání *dialog_option* na základě díla Dokonalý kód [8]. Funkce této metody bude popsána na ukázce metody v následujícím bloku.

```
1  else if (o is DefaultValue)
2  {
3      if (((DefaultValue)o).id_dialog_option_default.HasValue)
4      {
5          using (DBModel db = new DBModel())
6          {
7              string query = "SELECT '{group}' as ...";
8
9              option = db.Database.SqlQuery<dialog_ption>(query).First();
10             if (string.IsNullOrWhiteSpace(option.value))
11             {
12                 option.value = ((DefaultValue)o).value_default;
13                 option.Object = (DefaultValue)o;
14             }
15         }
16     }
17     else if (!string.IsNullOrWhiteSpace(((DefaultValue)o).value_default))
18     {
19         using (DBModel db = new DBModel())
20         {
21             string query = "SELECT '{group}' ...";
22
23             option = db.Database.SqlQuery<dialog_option>(query).First();
24             option.value = ((DefaultValue)o).value_default;
25         }
26     }
27 }
```

Metoda nejprve otestuje, jakého typu je vstupní parametr *o*.

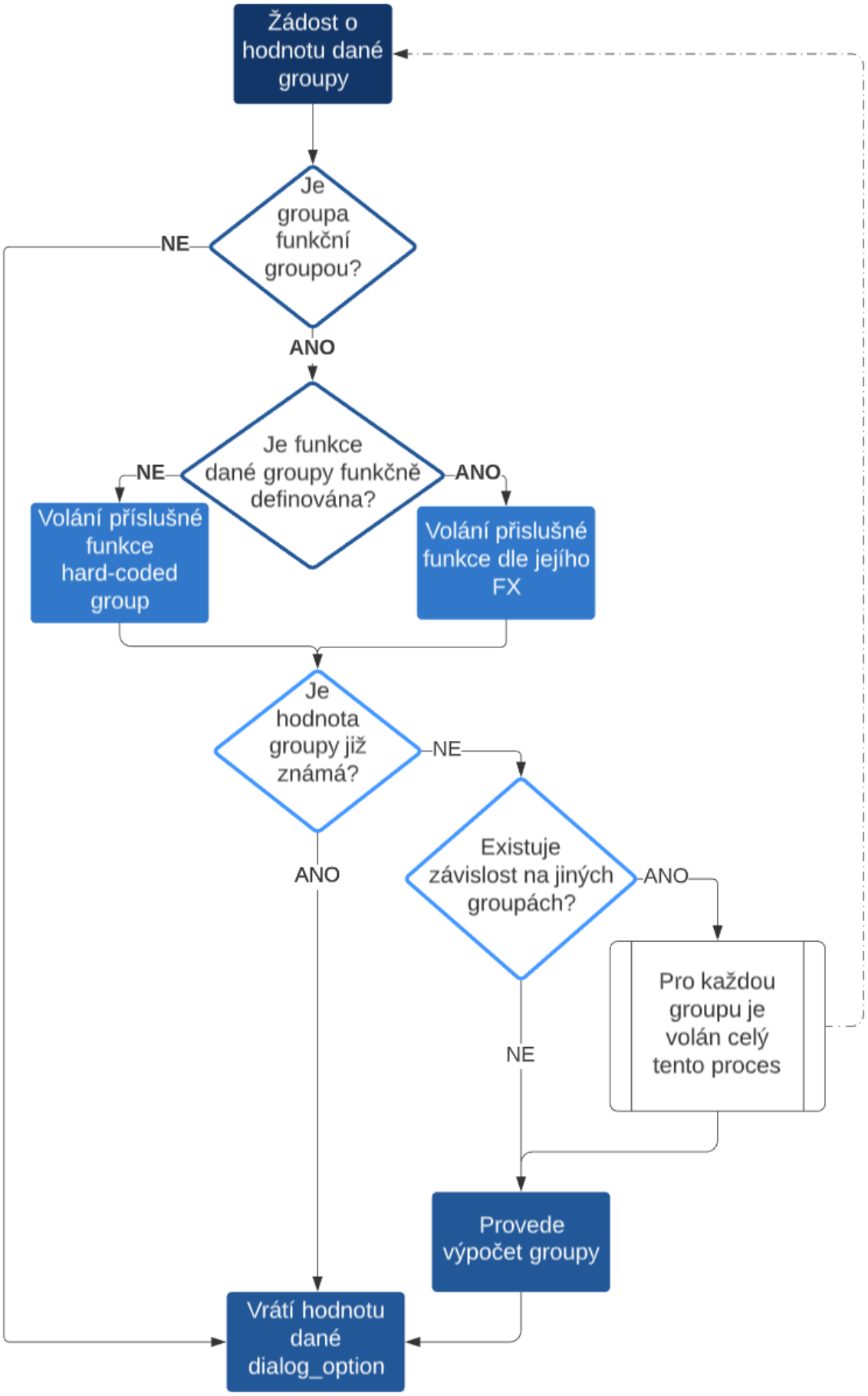
- dialog_option - již známá *dialog_option*
- DefaultValue - Objekt *DefaultValue*, který obsahuje informace o defaultní *dialog_option*
- DateTime? - datumové *groupy*, včetně *group* s neplatným vstupem
- int, long, string, double, bool - veškeré textové hodnoty bez bližších specifikací
- auto_data - komplexní informace o použitém voze

- GDV - komplexní informace o použité adrese
- PresetValue - objekt *PresetValue*, jež specifikuje bližší užití
- UsedPresetObject - podobně jako *PresetValue*
- UsedPresetUIDObject - pomocný objekt užívaný pro doplnění vypisovaných informací
- Range - objekt sloužící k dohledání příslušné *dialog_option*, která odpovídá rozsahu, kam patří daná hodnota

Následně dle typu metoda vyhledá odpovídající *dialog_option*, v tomto případě na základě *DefaultValue* buď vyhledá konkrétní *dialog_option* pomocí jejího *id*, nebo pomocí textové hodnoty. Po vyhodnocení daného typu je *dialog_option* nastavena na $used = 1$ a je přiřazena danému dialogu.

2.2.8 FunctionController

Třída **FunctionController** je rozcestníkem pro všechny funkce. Jejím hlavním úkolem je zjednodušit a centralizovat veškerou správu funkcí na jednom místě. Její hlavní metodou je metoda **ReturnCalculation(FunctionsVariables FunctionsVariables, long iFxCalling, TaskInfo taskInfo, long? iFxCalled)**. Vstupní parametry jsou již výše zmíněný objekt *FunctionVariables* (2.2.7), hodnota *groupy*, která se má vypočítat, taskInfo, což je rozšířený objekt *tasku*, a *groupa*, která si vyžádala přepočet této funkce. V samotném kódu se pak nacházejí dvě hlavní volání vedlejších metod, kde obě metody obsahují rozsáhlý *switch-case*. První metoda zjišťuje, je-li jméno funkce pro počítanou *groupu* známé, v tom případě hodnotu *groupy* vypočítá a zpracuje její výsledek. V opačném případě je volána druhá metoda, ve které je kódově specifikováno chování několika vyznamných *group* dle jejich id. Celý proces rekurzivního volání *group* je nastíněn na obrázku 2.8.



Obrázek 2.8: Digram rekurzivního volání výpočtu závislých *group*

2.3 Adaptační pravidla

V kapitolách výše bylo popsáno, jak funguje proces generování *masterů* a *klonů*. Vraťme se proto zpět k webovým formulářům. Jak bylo popsáno v kapitole 1.2.6, ty obsahují data o vozidle. Je zřejmé, že musí také obsahovat data o vlastníkovi vozidla, potažmo o pojistiteli. Zadávané informace jsou leckdy velice obsáhlé a obsahují informace také o členech rodiny, jejich věku či věku, kdy získali řidičský průkaz. V neposlední řadě také obsahují informaci o dni uzavření pojištění.

Budeme-li chtít tato naše vygenerovaná data jakkoli použít, musíme zaručit jejich aktuálnost, popřípadě jejich anonymitu. K tomu slouží metody **DateShift**(2.3.1), **Randomize**(2.3.4) a **VerifyCars**(2.3.3).

2.3.1 DateShift

DateShift je pravděpodobně nejsložitějším algoritmem celé aplikace. Jeho cílem je tzv. posunutí *dialogů* v datumové rovině. Budeme-li mít *dialog*, který jsme vytvořili například minulý měsíc, začátek pojištění bude pravděpodobně někdy v uplynulém měsíci. Budeme-li však chtít data využít dnes, žádný webový formulář nám to nedovolí, jelikož začátek pojištění musí vždy být v budoucnosti (výjimkou jsou některé subjekty, které umožňují pojistit již v den uzavření smlouvy). Pokud však posuneme pouze datum vzniku pojištění, a uděláme-li to vícekrát v průběhu roku, nebo uděláme-li to až příští rok, dostáváme se do situace, kdy data, která jsme vytvořili a která považujeme za nějaký dialog **ABC** s vozem **XYZ** a stářím jezdce **Q** rovným 18 let (většina datumů je však počítána s přesností na měsíce), již nejsou pravdivá, jak vidíme na obrázku 2.9.

VSTUPNÍ DATA			Stáří (měsíce)
Sjednání pojištění		15.3.2020	-
První registrace vozu		5.10.2005	173
Řidič 1	Datum narození	18.9.1992	329
	Datum vystavení ŘP	18.9.2010	113

VÝSTUPNÍ DATA (ke dni 15.4.2020)			Posun (dny)	Stáří (měsíce)
Sjednání pojištění		15.4.2020	31	-
První registrace vozu		5.10.2005	0	174
Řidič 1	Datum narození	18.9.1992	0	330
	Datum vystavení ŘP	18.9.2010	0	114

Obrázek 2.9: Nekonzistentnost dat při posunutí datumu sjednání pojištění

Naším cílem je tedy posunout všechny datumy. Zde však narážíme na řadu pravidel, která musíme dodržet. Datumy jsou rozděleny do čtyř skupin.

1. **Datum uzavření pojištění**

   Datum uzavření pojištění je specifický datum, pro který jediný platí, že musí být v budoucnosti a je vodící hodnotou pro celý *DateShift*. Platí pro něj následující:

   - Musí být v budoucnosti
   - Pokud vstupní datum je 1.1., pak výstupní datum musí být také 1.1.
   - v závislosti na typu *dialogu* je prováděn CheckDate (viz kapitola 2.3.2).

2. **Datumy volné (narozeniny,...)**

   Datumy volné jsou datumy, u kterých není prováděn CheckDate, jako jsou například narozeniny, které se neřídí žádnými pravidly.

3. **Datumy vázané (První registrace vozu, Datum vystavení řidičského průkazu,...)**

   Datumy vázané jsou všechny datumy, které jsou nějakým způsobem spjaty s jiným datumem, nebo se řídí například pracovními dny v týdnu. U těchto datumů je vždy prováděn CheckDate.

4. **Speciální pravidla**

   Mezi speciální pravidla patří například stáří vozu, které se odvíjí od jeho první registrace. Více v kapitole 2.3.3.

Pokud dodržíme všechna pravidla, která jsou ve zjednodušené formě popsaná výše, dostáváme konzistentní data, která vidíme na obrázku 2.10.

VSTUPNÍ DATA			Stáří (měsíce)
Sjednání pojištění		16.3.2020	-
První registrace vozu		5.10.2005	173
Řidič 1	Datum narození	18.9.1992	329
	Datum vystavení ŘP	18.9.2010	113

VÝSTUPNÍ DATA (ke dni 15.4.2020)			Posun (dny)	Stáří (měsíce)
Sjednání pojištění		16.4.2020	31	-
První registrace vozu		4.11.2005	30	173
Řidič 1	Datum narození	19.10.1992	31	329
	Datum vystavení ŘP	19.10.2010	31	113

Obrázek 2.10: Rovnost datumového rozdílu vstupních a výstupních dat při dodržení adaptačních pravidel

2.3.2 CheckDate

Jak vidíme na obrázku 2.10, posun datumu **první registrace vozu** je 30 dní, oproti ostatním posunům, jejichž hodnota je 31 dní. Za pomocí kalendáře lze snadno dohledat, že **5.11.2005** byla sobota. V tento den není možné pojistit vůz. Nejbližší den, kdy je možné pojistit vůz, je tak pátek, tedy **4.11.2005**. Pokud by na tento den vycházel nějaký svátek, musel by se datum posunout ještě dále, a to buď na 3.11.2005, nebo naopak na 7.11.2005. Vše za dodržení podmínky zachování datumového rozdílu (vždy ke dni sjednání pojištění) v měsících. Pokud by stále datum nebyl platným datumem (v tomto případě pracovním dnem), algoritmus by stejným způsobem pokračoval dále až k prvnímu platnému dni.

Pevné svátky, čili svátky s přesně daným datumem, jsou v kódu jako kolekce měsíce a dne, ke kterým se dopočítává aktuální rok. Pro výpočet pohyblivých datumů jsou použity funkce v bloku kódu níže.

```
1  private static DateTime feiertag(DateTime date)
2  {
3      while (FestFreitag(date.Year).Contains(date) || FreiFreitag(date.Year).
           Contains(date))
4          if (date.DayOfWeek == DayOfWeek.Saturday)
5              date = date.AddDays(2);
6          else
7              date = date.AddDays(1);
8          return date;
9  }
10 private static List<DateTime> FreiFreitag(int year)
11 {
12     var FreiFreitagList = new List<DateTime>();
13     var EasterSunday = EasterSundayCount(year);
14     FreiFreitagList.Add(EasterSunday.AddDays(-48));
15     FreiFreitagList.Add(EasterSunday.AddDays(-47));
16     FreiFreitagList.Add(EasterSunday.AddDays(-46));
17     FreiFreitagList.Add(EasterSunday.AddDays(-2));
18     FreiFreitagList.Add(EasterSunday.AddDays(+1));
19     FreiFreitagList.Add(EasterSunday.AddDays(+39));
20     FreiFreitagList.Add(EasterSunday.AddDays(+50));
21     FreiFreitagList.Add(EasterSunday.AddDays(+60));
22     return FreiFreitagList;
23 }
```

2.3.3 VerifyCars

Výše zmíněná **první registrace vozu** nám zde posloužila jako příklad datumu, který by bez použití *CheckDate* nebyl validním. V reálných datech ovšem nese ještě jednu zásadní informaci. Naším vozem pro tuto ukázkovou situaci může být například vůz *VOLVO S60 R 2.5 AWD* z roku 2004. Tento vůz sice nebyl vyroben

v roce 2005, nicméně je možné koupit vůz, který byl vyroben v loňském roce, a pojistit ho letos. Pokud bychom ale takový vůz, tedy vůz zakoupený v roce 2020, vyrobený v roce 2019, kdy byla ukončena jeho výroba, chtěli pojistit v roce 2021, většina subjektů nám to již neumožní. Stejný případ by nastal i u našeho vozu Volvo, kdybychom náš ukázkový *dialog* chtěli použít za dva měsíce. První registrace vozu by tak byla v lednu roku 2006, což je 2 roky od ukončení výroby našeho vozu.

Metoda **VerifyCars** se stará o dohledání a nahrazení již nevalidního vozu vozem validním. Vyhledávací algoritmus se snaží dohledat co nejpodobnější vůz, a to na základě cenového rozpětí, stejného typu pohonné jednotky (benzín, diesel, hybrid,...) a podobnostního faktoru, který vzniká multiplikací tří parametrů, které by se daly popsat jako rizikové třídy vozu pro pojištění.

2.3.4 Randomize

Metoda **Randomize**, jak již její název napovídá, umožňuje randomizaci dat. Ta je trojího způsobu:

1. **Randomizace datumů**

   Umožňuje při provádění *DateShiftu*, který probíhá nastavením výstupního datumu začátku pojistění, všechny posouvané datumy začátku pojištění randomizovat v určitém rozsahu, který lze nastavit v GUI. Stejně tak je možné nastavit rozsah randomizace pro ostatní datumy. Tato randomizace vždy probíhá až po posunu dat.

2. **Randomizace adres**

   Podobně jako je tomu u nahrazování vozů popsaném v kapitole 2.3.3, probíhá randomizace adres na základě několika parametrů určujících podobnost dané adresy.

3. **Randomizace aut**

   Probíhá stejným způsobem jako při nahrazování nevalidních aut.

2.4 Ostatní funkcionalita aplikace

Aplikace je uceleným nástrojem pro práci s daty popsanými v kapitole 1.2, především pro data z databázových tabulek *dialog_option* a *dialog*. Z toho důvodu bylo do aplikace implementováno několik dalších funkcí, které usnadňují především verifikaci dat, popřípadě zjednodušené klonování dat.

1. **Dialog Comparer**

   Dialog comparer nalezneme mezi záložkami aplikace a umožňuje nám porovnání dvou *dialogů*. Data se vypisují ve formátu otázka - *dialog* A - hodnota A - *dialog* B - hodnota B. Pro lepší přehlednost je zde možnost specifikovat poprovnávané *groupy*.

2. **Soft clone**

   Stejně jako dialog comparer nalezneme i Soft clone mezi záložkami aplikace. Tato možnost umožňuje vytvořit kopie zadaných *dialogů* se specifikovanou změnou, tedy záměnou *dialog_option* za jinou *dialog_option*.

3 Dokumentace

Dokumentace této aplikace je trojího způsobu, tak jak bylo dohodnuto se zadavatelskou firmou.

1. **Samopopisný kód**

   Jak je zřejmé z ukázek kódu výše, aplikace byla psaná s co možná největším důrazem na užití popisných názvů, ať již tříd, metod či proměnných. V kódu samotném se pak nacházejí drobné komentáře, které dovysvětlují některé možné nesrovnalosti.

2. **Samotný text diplomové práce**

   V případě začleňování nového programátora do oblasti *dialog - groupy - dialog_option* bude velmi přínosný samotný text diplomové práce, který vysvětluje jednotlivé databázové souvislosti týkající práce s těmito daty. Stejně tak je tomu v případě předání kódu aplikace Edittool další osobě pro pozdější úpravy.

3. **Dropbox paper**

   Podrobnější dokumentace některých částí, jako jsou hlavní *helpery* aplikace, všechny výpočetní funkce, které byly implementovány, a některé hlavní třídy, je zpracovávána v online podobě v aplikaci **Dropbox paper**. Vzhledem k citlivosti dat zde popsaných je však tato dokumentace neveřejným dokumentem.

Při hledání možností, jak jednoduše a efektivně psát dokumentaci se jako nejlepší volba ukázala právě aplikace Dropbox paper. Uživatelské prostředí aplikace působí velice dobře. Díky přednastaveným zkratkám, ať již klávesovým či kódovým, je možné psaní zefektivnit a zpřehlednit. Možnosti editace nejsou rozsáhlé, přesto jsem je shledal jako dostatečné. Možnost sbalení jednotlivých kapitol vítám, simuluje práci s kódem stejně jako možnost vkládání ukázek kódu. Barevné označování se při systematickém užití ukázalo jako velice přehledné. Hlavní výhodou je možnost exportovat data ve třech formátech, a to .pdf, .docx a .md, což je běžně užívaný formát pro psaní dokumentace.

4 Závěr

Vzhledem ke stále se obměňujícím datům, ať již ve formě dat vstupních či požadovaných dat výstupních, je aplikace velmi živou a neobejde se bez dalších zásahů. Aplikace byla psána s důrazem na okamžitou funkčnost s postupně přidávanou funkcionalitou. Testování proto často probíhalo v přímém nasazení aplikace a veškeré nesrovnalosti byly v co nejkratší době odladěny. V současné chvíli aplikace splňuje vše, co bylo jejím zadáním. Výkon aplikace, respektive schopnost generovat data, by se dnes dal považovat jako 10 000x - 100 000x větší, než-li tomu bylo před jejím zavedením při parciálně strojovém zpracování za použití lidského elementu při vytváření dat. Celkově se pak bavíme o mnohem větším přínosu, který plyne z implementace verifikačních a adaptačních funkcí. Kvůli aplikaci přibylo i velké množství režie, která ovšem vyplývá z množství zpracovávaných dat a pravidel k tomu potřebných. Aplikace vypočítává přes 3700 funkčních *group* pomocí 51 definovaných funkcí s přibližně 80 různými způsoby volání, to vše pro dvě až tři desítky různých zdrojů, také využívá přes 13000 konvertovacích pravidel. To vše nad daty, jejichž hlavní databázová tabulka obsahuje téměř 800 milionů záznamů. Správnost výstupních dat ve velké míře závisí i na datech vstupních. Vzhledem k velkému množství zpracovávaných dat z nejrůznějších zdrojů, které jsou rovněž obměňovány, nelze zaručit stoprocentní správnost všech vstupních dat, jednak těch přímo zpracovávaných při generování nových dat, či dat spjatých s režií a chodem celé aplikace. Jsou-li však tato data v ideálním stavu (tato data nespadají pod aplikaci EditTool a není možné zaručit jejich správnost), pracuje aplikace přesně a s korektními výsledky a je denně používaným nástrojem pro editaci, modifikaci i transformování dat. Největším úskalím celé aplikace byly kalendářní posuny s ohledem na omezující podmínky. Tato část se při vývoji aplikace ukázala jako nejsložitější a nejnáchylnější na jakékoli změny vstupních dat. Postupem času však bylo dosaženo téměř stoprocentního výsledku. Nedokonalost vyplývá z množství dat, jež jsou tímto způsobem adaptována, popřípadě z množství způsobů, kterými jsou datumová posunutí používána. Verifikování těchto posunů nad velkými sadami dat je velmi obtížné

a drobné nedostatky se tak odhalují pomalu. V aktuálním stavu však veškeré posuny fungují a není znám žádný případ chybného posunutí.

Použitá literatura

[1] *Co je to databáze MySQL?* [online]. Oracle Česká republika, 2020 [cit. 28. 05. 2020]. Dostupné z: https://www.oracle.com/cz/database/what-is-database.html.

[2] *Entity Framework 6* [online]. Microsoft, 2020 [cit. 28. 05. 2020]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/ef6/.

[3] *Introducing Telerik® Data Access* [online]. Telerik® Data Access by Progress, 2020 [cit. 28. 05. 2020]. Dostupné z: https://docs.telerik.com/data-access/data-access-introduction.

[4] *WinfoBOT dokumentace*. 1. vyd. Liberec: Interní dokument, 2018.

[5] RUD, Olivia Parr. *Data Mining: praktický průvodce dolováním dat pro efektivní prodej, cílený marketing a podporu zákazníků (CRM)*. Praha: Computer Press, 2001. ISBN 80-722-6577-6.

[6] FRIEDL, Jeffrey E. F. *Mastering regular expressions*. Sebastapol, CA: O'Reilly, 2006. ISBN 0596528124.

[7] ROBERTS, Jason. *Clean C: Readable, Maintainable, Pleasurable C*. Jason Roberts, 2015. e-kniha.

[8] MCCONNELL, Steve. *Dokonalý kód: umění programování a techniky tvorby software*. Brno: Computer Press, 2005. ISBN 80-251-0849-X.