

Vizualizace stavu obráběcích strojů Designer rozmístění strojů

Visualization of Machine Tools at Production Halls
Layout Designer

Bc. Radomír Sohlich

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Radomír SOHLICH**
Osobní číslo: **A10845**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Vizualizace stavu obráběcích strojů ve výrobní hale -
Designer rozmístění strojů**

Zásady pro vypracování:

1. Seznamte se se současným způsobem tvorby layoutů.
2. Na základě požadavků zadavatele navrhnete aplikaci pro tvorbu layoutů.
3. Implementujte aplikaci na bázi .NET Frameworku a programovacího jazyka C Sharp.
4. Pro návrh uživatelského rozhraní zvolte vhodnou technologii v rámci .NET Frameworku.
5. V aplikaci implementujte připojení ke stávající databázi.
6. Ve spolupráci se zadavatelem aplikaci otestujte.
7. Aplikaci nasadte do reálného provozu.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. JOHNSON, Glenn. MCTS Self-Paced Training Kit (Exam 70-516): Accessing Data with Microsoft? .NET Framework 4. Redmond, Washington 98052-6399: Microsoft Press, 2011. ISBN 978-0-7356-6260-0.
2. STOECKER, Matthew. MCTS self-paced training kit (Exam 70-511): Windows applications development with Microsoft .net Framework 4. Redmond, WA: Microsoft Press, 2011. ISBN 978-073-5627-420.
3. MACDONALD, Matthew. Pro WPF in C-Sharp 2010: Windows presentation foundation in .NET 4. New York, N.Y.: Distributed to the book trade worldwide by Springer-Verlag, 2010, 1181 s. Experts voice in .NET. ISBN 14-302-7205-8.
4. MICROSOFT. MSDN Library [online]. 2012 [cit. 2012-01-17]. Dostupné z: <http://msdn.microsoft.com/en-us/library>
5. SKEET, Jon. C-Sharp in depth. 2nd ed. Stamford, CT: Manning, 2011, 554 s. ISBN 19-351-8247-1.
6. DEITEL, Paul J, Harvey M DEITEL a Paul J DEITEL. Visual C-Sharp 2010: how to program. 4th ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2011, 959 s. How to program series. ISBN 01-321-5142-1.

Vedoucí diplomové práce:

Ing. Petr Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlíně dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.

děkan



doc. Mgr. Roman Jašek, Ph.D.

ředitel ústavu

ABSTRAKT

Práce obsahuje analýzu dosavadní tvorby podkladů pro vizualizaci výrobních hal. Na základě podkladů od zadavatele je navržena a implementována aplikace pro zjednodušení tvorby podkladů. V posledních fázích je aplikace testována a rozvíjena na základě dodatečných požadavků. Výsledkem práce je úspěšné uvedení aplikace do reálného provozu.

Klíčová slova: *design aplikace, vizualizace, WPF, LINQ*

ABSTRACT

A thesis contains analysis of existing layout creation. On base of submitter the application for easy layout creation is designed and implemented. A thesis includes development of application, based on additional functionality requests, and testing. Final stage is deployment of application.

Keywords: *application design, visualization, WPF, LINQ*

Poděkování:

Poděkování patří panu Ing. Petrovi Šilhavému, Ph. D, za odborné vedení diplomové práce, zejména při zpracování teoretické části. V neposlední řadě bych chtěl poděkovat IT oddělení společnosti Meopta – optika, s.r.o za spolupráci při testování aplikace.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

OBSAH	11
ÚVOD.....	15
I. TEORETICKÁ ČÁST	17
1 ÚVODNÍ STUDIE.....	18
1.1 POZADÍ VZNIKU APLIKACE	18
1.2 ÚVOD DO PROBLEMATIKY	19
1.3 STÁVAJÍCÍ PROSTŘEDKY A ZABEZPEČENÍ.....	20
1.3.1 SQL INJECTION	21
1.4 ANALÝZA POŽADAVKŮ	21
1.4.1 ROZDĚLENÍ POŽADAVKŮ	22
1.5 PŘÍPADY UŽITÍ.....	23
1.5.1 SCÉNÁŘE PRO PŘÍPADY UŽITÍ.....	24
1.5.1.1 UC001 - Vytvoření nového shape v databázi	24
1.5.1.2 UC003 - Editace stávajícího shape	25
1.5.1.3 UC005 - Vytvoření nového layoutu	26
1.5.1.4 UC008 - Uložení layoutu do souboru	27
1.5.1.5 UC011 - Načtení layoutu ze souboru	28
1.5.1.6 UC013 - Uložení layoutu do databáze	28
1.5.1.7 UC015 - Editace stávajícího layoutu.....	29
1.5.1.8 UC016 - Nalezení layoutu v databázi	29
1.5.1.9 UC018 - Odstranění layoutu z databáze.....	30
1.5.1.10 UC019 - Duplikování layoutu v databázi.....	30
1.6 VÝVOJOVÁ METODA - PROTOTYPING	31
2 ANALÝZA TECHNOLOGIÍ.....	32
2.1 .NET.....	32
2.1.1 TECHNOLOGIE PRO TVORBU GUI	33
2.1.2 TECHNOLOGIE PRO PŘÍSTUP K DATABÁZI	34
2.1.2.1 LINQ to SQL.....	34
2.1.3 TECHNOLOGIE PRO PŘÍSTUP K XML	35
II. PRAKTICKÁ ČÁST	37
3 DESIGN.....	38
3.1 NÁVRH GUI.....	38
3.2 NÁVRH DATOVÝCH STRUKTUR.....	41
3.2.1 POŽADAVKY NA ATRIBUTY JEDNOTLIVÝCH OBJEKTŮ	41
3.2.2 RELAČNÍ MODEL A TVORBA DATABÁZOVÉ STRUKTURY	42
3.2.3 IO INTEGRITNÍ OMEZENÍ	44
3.3 NÁVRH FUNKCIONALITY UNDO/REDO	44
4 PRŮBĚH VÝVOJE	46
4.1 VOLBA TECHNOLOGIÍ	46
4.2 INICIALIZAČNÍ PROTOTYP	46
4.3 PRVNÍ EVOLUČNÍ CYKLUS	47

4.4	DRUHÝ EVOLUČNÍ CYKLUS	48
4.5	TŘETÍ EVOLUČNÍ CYKLUS	50
5	FINÁLNÍ PRODUKT	52
5.1	POPIS APLIKACE	52
5.2	TŘÍDY GUI.....	53
5.2.1	MAINWINDOW	53
5.2.1.1	Načtení Shape do nabídky.....	54
5.2.1.2	Implementace grafické editace layoutu.....	55
5.2.1.3	Implementace tabulkového pohledu na layout.....	59
5.2.1.4	Zobrazování chyb a jejich zjišťování.....	61
5.2.1.5	Implementace Undo/Redo.....	62
5.2.1.6	Implementace Snap to grid a funkce ORTHO	63
5.2.1.7	Implementace otevírání/ukládání do souboru	63
5.2.1.8	Implementace otevírání/ukládání layoutu do databáze	64
5.2.1.9	Implementace funkcionality AutoSave	64
5.2.1.10	Klávesové zkratky	65
5.2.2	LAYOUTBROWSERDIALOG A SHAPEBROWSERDIALOG.....	66
5.2.3	<i>LAYOUTMNGR, SHAPEMNGR, TEXTMNGR A LAYOUTITEMMNGR</i>	<i>66</i>
5.2.4	PREFERENCESDIALOG	67
5.3	TŘÍDY DATA & LINQ	67
5.3.1	LAYOUTCLASS	67
5.3.2	SHAPECLASS	68
5.3.3	LAYOUTITEM	68
5.3.4	LAYOUTTEXT.....	69
5.4	TŘÍDY HELPER.....	69
5.4.1	DATASOURCECLASS	69
5.4.2	ERRORITEM.....	70
5.4.3	AUTOSAVE.....	70
5.4.4	LIMITEDSTACK.....	70
5.4.5	CLIPBOARD	70
5.4.6	SNAPGRID.....	71
5.5	LOKALIZACE APLIKACE.....	72
6	DEPLOYMENT APLIKACE	73
6.1	TVORBA UŽIVATELSKÉHO MANUÁLU	73
6.1.1	ZAMĚŘENÍ UŽIVATELSKÉHO MANUÁLU	73
6.1.2	KONCEPCE MANUÁLU.....	73
6.2	INSTALAČNÍ BALÍČEK	73
6.2.1	VYTVOŘENÍ INSTALAČNÍHO BALÍČKU MSI.....	74
6.2.2	OTESTOVÁNÍ INSTALAČNÍHO BALÍČKU	74
	ZÁVĚR	75
	ZÁVĚR V ANGLIČTINĚ.....	77
	SEZNAM POUŽITÉ LITERATURY.....	79
	CITOVANÁ LITERATURA	81

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	83
SEZNAM OBRÁZKŮ	85
SEZNAM TABULEK.....	87
SEZNAM PŘÍLOH.....	89

ÚVOD

Obecným cílem této práce je využít nabyté teoretické znalosti z akademického prostředí pro vytvoření praktického fungujícího projektu, který není samostatnou aplikací, nýbrž jsou na něj vázány další části většího celku, a tedy je nutné, se flexibilně přizpůsobovat případným vlivům a situacím, které do oblasti projektu zasahují.

Jedním z hlavních podnětů k tomuto tématu byla možnost získat cenné zkušenosti nejen na poli programátorském, ale i na poli metodik a firemních postupů. Projitím celého procesu vývoje aplikace, od návrhu po samotnou implementaci, si vytvořit „dobré návyky“ z hlediska analýzy požadavků, jejich specifikace a návrhu. Při samotné implementaci se naučit řadu takzvaných „best practice“ řešení a prohloubit znalosti programovacího jazyka C# a technologií frameworku .NET .

V teoretické části práce je uvedena úvodní studie, do které patří i analýza prostředí, ve kterém bude aplikace vyvíjena, popis jednotlivých vazeb na toto prostředí a souvislosti s výrobou probíhající v organizaci. Součástí úvodní studie je i popis projektu organizace, jehož je aplikace součástí a jednotlivé fáze tohoto projektu. Úvodní studie zahrnuje také analýzu technických prostředků a technologií, využitelných pro tvorbu aplikace. Jednou z hlavních částí úvodní studie je analýza požadavků na aplikaci a rozpracování jednotlivých scénářů případů užití. Studie zahrnuje i popis použité metodiky pro vývoj aplikace a klade si za cíl připravit podklady pro samotný návrh aplikace a její úspěšnou realizaci.

Praktická část zahrnuje návrh aplikace od její grafické podoby po návrh datových struktur. Součástí návrhu aplikace jsou i principy některých funkcionalit a popis jejich chování. Účelem návrhu je vytvořit teoretické jádro, podle kterého bude aplikace implementována. Samotná implementační část pak přináší popis vývojového cyklu aplikace a v konečné fázi popis hlavních částí jednotlivých tříd a funkcionalit tak, jak byly realizovány v kódu. Závěrem praktické části jsou popsány aktivity spojené s nasazením aplikace do provozu.

I. TEORETICKÁ ČÁST

1 ÚVODNÍ STUDIE

Před samotným návrhem aplikace je nutné analyzovat současný stav a požadavky zadavatele na výslednou aplikaci a vytvořit si tak základní představu o jejím návrhu. Tento krok je nutný zejména k efektivnímu návrhu, aby v pozdějších fázích nedocházelo ke zbytečně náročným úpravám (v komerčním prostředí i nákladným) nebo v krajním případě k selhání celého projektu.

1.1 Pozadí vzniku aplikace

ERP systémy jsou v současné době nutností pro řízení každé střední až velké organizace zabývající se výrobní činností. Jejich cílem je zjednodušit lidem práci, přinést větší přehlednost a efektivnost prováděných procesů a operací, automatizovat některé procesy a vyhnout se tak pochybení lidského faktoru. Těchto systémů se na trhu nachází velké množství, řada z nich je velmi komplexních, propracovaných a poskytují tzv. modulární architekturu. Ta umožňuje zákazníkovi výběr modulů dle jeho přání a potřeb.

Moduly jsou navrženy obecně pro použití v jakémkoliv odvětví výroby, a tedy v některých případech může tato obecnost přinést komplikace v nasazení ERP na již existující výrobu.

Proto se někdy firmy rozhodují k implementaci vlastního informačního systému, který bude tvořen na míru jejich požadavkům. Toto rozhodnutí může současně s výhodou přesně padnoucího řešení přinést v delším časovém horizontu i finanční úsporu v porovnání s nasazením „externího“ informačního systému, jehož provoz může být podmíněn zakoupení drahé licence.

Organizace **Meopta – optika, s. r. o.** se rozhodla jít cestou návrhu a implementace vlastního informačního systému. Tím se zabývá interní projekt organizace s názvem **Shop Floor Control – Machine monitoring** (dále jen SFC).

Tento projekt je součástí dlouhodobé koncepce **Meopta Improvement Program**, která je realizována napříč výrobními divizemi a klade si za cíl zefektivnit práci a zároveň přinést větší přehlednost a sledovatelnost o procesech probíhajících v organizaci.

Projekt SFC se nyní nachází v druhé fázi vývoje. První fáze vývoje proběhla v období leden – duben 2011. V této fázi byla vybraná výrobní zařízení osazena monitorovacími prvky a vytvořena aplikace pro plánování výroby na konkrétní stroje. Zároveň byla vytvořena jednorázová aplikace pro vizualizaci ručně zadávaných dat přicházejících

z monitorovacích zařízení (Obrázek 1) a celý systém spuštěn ve zkušebním provozu bez přímého propojení do firemního IS.



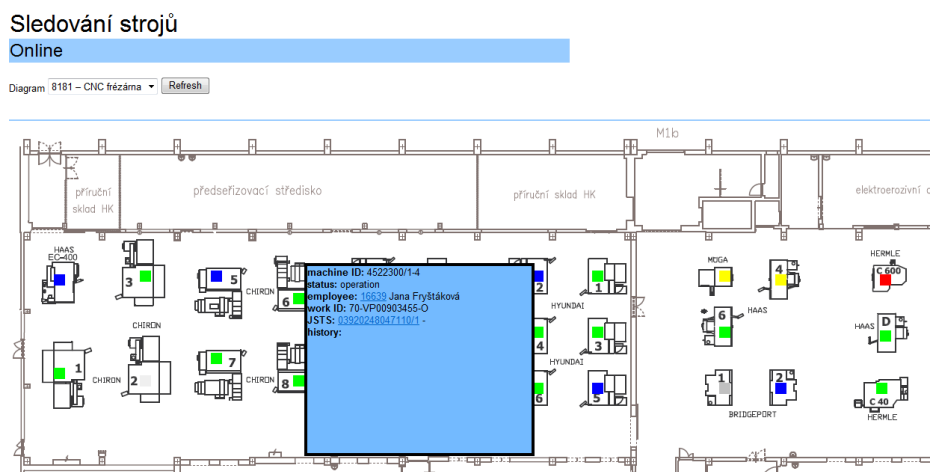
Obrázek 1: Ukázka monitorovacího zařízení

V druhé fázi projektu je plánováno zapojení dalších výrobních hal do projektu, sběr reálných dat (automaticky snímaných), rozšíření vizualizace na firemní intranet a tvorba podpůrných modulů.

1.2 Úvod do problematiky

Jak jsem již zmiňoval, rozšíření projektu na další výrobní haly a tedy i potřeba jejich vizualizace si vyžádala další tvorbu vizualizačních podkladů (tzv. *layoutů*).

Původní tvorba podkladu realizovaná v první fázi projektu SFC zahrnovala vytvoření výkresového podkladu výrobní haly v CAD systému AutoCAD (včetně rozmístění strojů) a následné vytvoření dynamické webové stránky s ručně pozicovanými vizualizačními elementy v podobě barevných obdélníků indikujících činnost stroje (Obrázek 2).

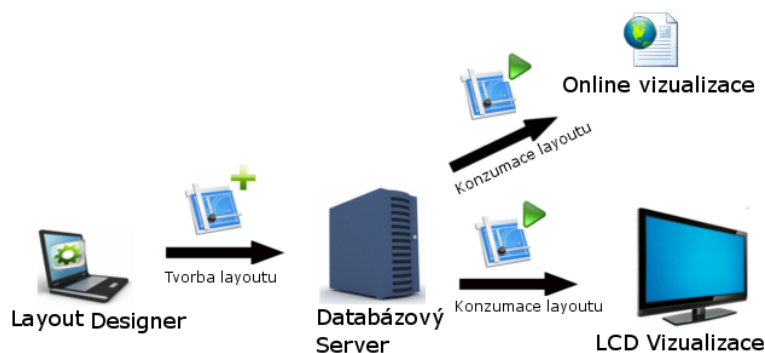


Obrázek 2: Ukázka stávajícího ručně tvořeného *layoutu*

Tato činnost samozřejmě není jednorázová, jelikož rozmístění strojů se mění na základě požadavků momentální zakázky. Podklady se tedy budou v delších časových horizontech, řádově půl roku, měnit.

Je zřejmé, že tato činnost není efektivní a dochází zde k plýtvání času zaměstnanců, jejichž odbornost nemá s plánováním výroby co dočinění. Proto vznikla potřeba vytvořit specializovaný nástroj umožňující jednoduchou tvorbu grafického podkladu a definici vazeb mezi reálnými stroji, který by do budoucna přinesl organizaci úsporu pracovních sil, času a delegoval činnost tvorby podkladů pro vizualizaci na patřičné profese, tedy na management a mistry na výrobních halách.

Uvažovaný koncept je tedy vytvoření aplikace pro tvorbu, návrh a správu *layoutů*, která je tématem této diplomové práce. Na tuto aplikaci navazuje webová aplikace pro online vizualizaci a vizualizaci na LCD obrazovkách viz schéma (Obrázek 3).



Obrázek 3: Schéma systému *layoutů* v projektu Shop Floor Control

1.3 Stávající prostředky a zabezpečení

V organizaci je aktuálně v provozu databázový server Microsoft SQL Server 2008 R2, na kterém je umístěna databáze celého informačního systému. V tomto případě to znamená bezproblémové nasazení nejen technologií .NET pro práci s databází, ale i administračních nástrojů, které jsou součástí Visual Studio 2010.

Pro přístup do podnikové sítě je využíváno služby *Active Directory*, každý uživatel se tedy hlásí do domény a jeho iniciály jsou ověřovány na podnikovém serveru. Pro přístup k databázi je využíváno tzv. *Windows Authentication*, kdy pro přístup k databázi není v *Connection String* uvedeno uživatelské přihlašovací jméno a heslo. Místo toho jsou

využity iniciály od aktuálně přihlášeného uživatele v systému *Windows*. *Connection String* tedy není třeba nějak dodatečně zabezpečovat šifrováním.

1.3.1 SQL Injection

V obecných případech je zabezpečení proti *SQL injection* nutno řešit explicitně v rámci aplikace. A to formou nahrazování znaků tzv. *Escape Sequences* (nahrazení speciálních znaků kombinací zpětného lomítka a písmene nebo řetězce) a jiných opatření. V tomto případě bude použit komponent *LINQ to SQL*, který zabraňuje tomuto způsobu útoku tím, že používá *SqlParameter*. To znamená, že před všechny vkládané proměnné je aplikován tzv. *verbatim string literal*, který odebírá jakýkoliv funkční význam znaku v řetězci. Při správném použití je tedy *LINQ to SQL* vůči *SQL injection* bezpečný.

1.4 Analýza požadavků

Layout Designer je aplikace, řešící tvorbu propojení *výkresových podkladů rozmístění strojů a metadat pro realtime vizualizaci výroby* (dále jen *layout*). Aplikace musí být navržena tak, aby poskytovala všechny dnes standardní funkce pro uživatelsky pohodlnou práci (tzv. *user-friendly UI*). Je vyžadováno intuitivní GUI pro tvorbu *layoutu*. Zároveň je požadována možnost opětovného využití *layoutů* a jejich další upravitelnost.

Aplikace musí komunikovat se stávající databází. Musí zajistit ukládání a správu *layoutů* včetně *grafických prvků pro jejich tvorbu* (dále jen *shape*) na databázový server organizace. Při nedostupnosti databázového serveru musí aplikace umožňovat offline tvorbu *layoutů* a jejich ukládání do souboru, jejich následné načtení a v online režimu nahrání *layoutu* ze souboru do databáze. V rámci správy databáze musí aplikace dohlížet na dodržování integritních omezení specifikovaných níže v dokumentu.

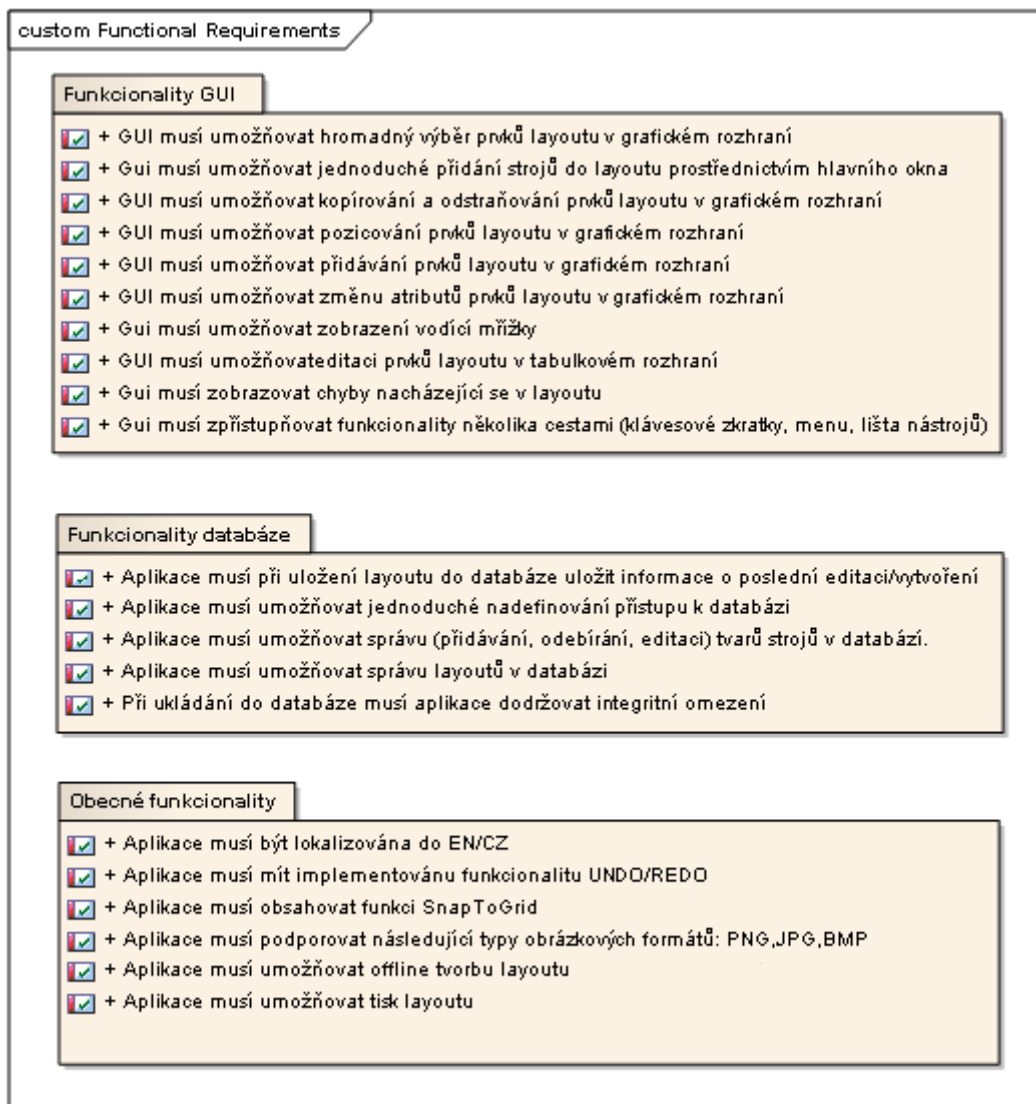
V uloženém *layoutu* musí být k dispozici informace o jeho platnosti, poslední změně a informace o uživateli, který *layout* naposledy změnil. Toto platí i o databázi *shape*. Designer bude umožňovat rozlišení *layoutu* dvojího typu, a to pro intranetové rozhraní a pro zobrazení na LCD na výrobních halách.

Aplikace musí být schopna použít pro výkresový podklad *layoutu* a grafický podklad pro *shape* obvyklé typy obrazových formátů specifikovaných dále v dokumentu.

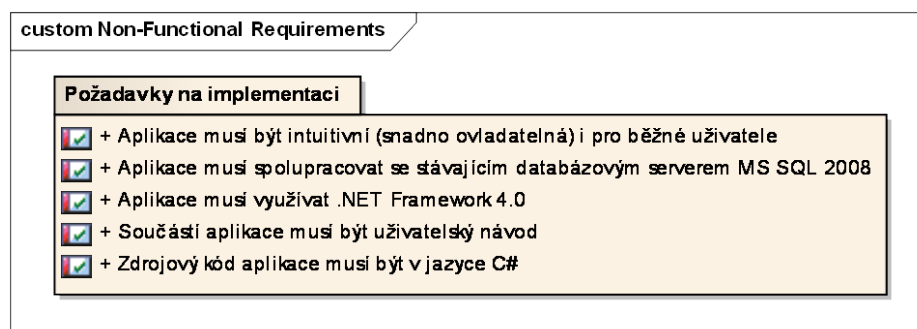
Implementační platforma pro tvorbu aplikace musí být založena na .NET Frameworku verze 4.0 a zdrojový kód psaný v jazyce C#.

1.4.1 Rozdělení požadavků

Pro přehlednost je dobré si požadavky rozdělit do skupin a při nejasnostech požadavky dále specifikovat.



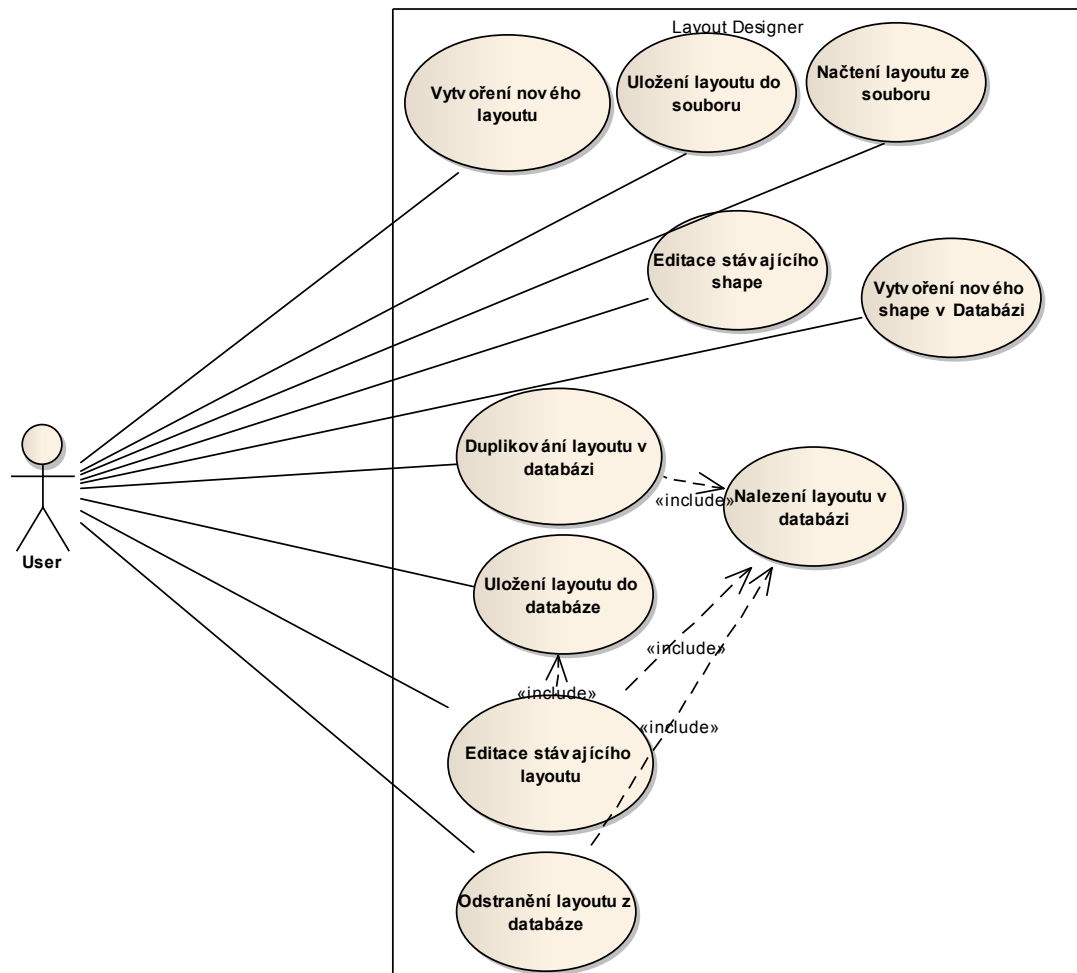
Obrázek 4: Package diagram funkčních požadavků



Obrázek 5: Package diagram nefunkčních požadavků

1.5 Případy užití

Pro ujasnění funkcionalit a ověření, že nebyla opomenuta žádná záležitost, byl vytvořen *use - case model* (Obrázek 6).



Obrázek 6: Use Case Model pro Layout Designer

Identifikace UC	Název
UC001	Vytvoření nového <i>shape</i> v databázi
UC003	Editace stávajícího <i>shape</i> v databázi
UC005	Vytvoření nového <i>layoutu</i>
UC008	Uložení <i>layoutu</i> do souboru
UC011	Načtení <i>layoutu</i> ze souboru
UC013	Uložení <i>layoutu</i> do databáze
UC015	Editace stávajícího <i>layoutu</i>
UC016	Nalezení <i>layoutu</i> v databázi
UC018	Odstranění <i>layoutu</i> z databáze
UC019	Duplikování <i>layoutu</i> v databázi

Tabulka 1: Přehled primárních případů užití a jejich identifikace

1.5.1 Scénáře pro případy užití

Jednotlivé případy užití budou podrobně popsány v následujících sekcích. V tomto případě je důležité navrhnout alternativní scénáře reagující na nestandardní situace.

1.5.1.1 UC001 - Vytvoření nového shape v databázi

Cíl případu užití	Vytvořit nový záznam pro reprezentaci typu stroje v podnikové databázi.		
Vstupní podmínky	Uživatel má k dispozici grafiku stroje ve formátu JPG,BMP nebo PNG.		
Výstupní podmínky	Je vytvořen nový záznam v databázi.		
Základní scénář	Krok	Role	Akce
	1	System	zobrazí dialog pro vytvoření shape.
	2	Aktér	zadá povinné údaje o typu stroje, a to: název,popis, grafiku a data platnosti
	3	System	provede převedení podkladu do příslušného formátu
	4	Aktér	zvolí, že chce shape uložit.
	5	System	provede kontrolu zadaných údajů.
	6	System	pokud není chyba uloží záznam. pokud je chyba UC002 - alternativní scénář - chyba vstupních údajů.
Alternativní scénář UC002	Krok	Role	Akce
	6a1	System	zobrazí chybové hlášení, že shape nelze vložit do databáze.
	6a2	System	nabídne volbu pro zrušení uložení a pro opravu údajů. (Pokud aktér zvolí opravu údajů, pokračuje se krokem 2 hlavního scénáře)

1.5.1.2 UC003 - Editace stávajícího shape

Cíl případu užití	Editovat záznam typu stroje v podnikové databázi.		
Vstupní podmínky	Uživatel je připojen k databázi.		
Výstupní podmínky	Záznam o typu stroje je v databázi editován.		
Základní scénář	Krok	Role	Akce
	1	System	zobrazí dialog pro procházení <i>shape</i> .
	2	Aktér	vybere <i>shape</i> pro editaci
	3	System	zobrazí dialog pro editaci <i>shape</i>
	4	Aktér	změní atributy záznamu
	5	System	provede kontrolu zadaných údajů.
	6	System	pokud není, chyba uloží záznam. pokud je chyba UC004 - alternativní scénář - chyba vstupních údajů.
Alternativní scénář	Krok	Role	Akce
	6a1	System	zobrazí chybové hlášení, že <i>shape</i> nelze vložit do databáze.
	6a2	System	nabídne volbu pro zrušení uložení a pro opravu údajů. (Pokud aktér zvolí opravu údajů, pokračuje se krokem 2 hlavního scénáře)

1.5.1.3 UC005 - Vytvoření nového layoutu

Cíl případu užití	Vytvořit nový záznam <i>layoutu</i> v podnikové databázi.		
Vstupní podmínky	Uživatel má grafický podklad pro <i>layout</i> a požadované atributy.		
Výstupní podmínky	Je vytvořen <i>layout</i> pro uložení.		
Základní scénář	Krok	Role	Akce
	1	Aktér	provede volbu pro vytvoření nového <i>layoutu</i>
	2	Systém	založí nový <i>layout</i> . Pokud je již aktivní jiný <i>layout</i> - UC006 - alternativní scénář 1 - neuložený <i>layout</i>
	3	Aktér	provede vložení podkladu a zadání atributů <i>layoutu</i> .
	4	Systém	převéde podklad do <i>layoutu</i> , aktualizuje údaje <i>layoutu</i> . Pokud chyba v souboru grafiky - UC007 - chyba souboru
	5	Aktér	přidá stroje do <i>layoutu</i>
Alternativní scénář UC006	Krok	Role	Akce
Neuložený <i>layout</i>	2a1	Systém	zobrazí varovné hlášení, že je aktivní jiný <i>layout</i>
	2a2	Aktér	zvolí uložení nebo zahození změn a pokračuje se krokem 2 v hlavním scénáři
Alternativní scénář UC007	Krok	Role	Akce
Chyba souboru	4a1	Systém	zobrazí chybové hlášení, že je soubor nečitelný nebo špatného formátu
	4a2	Aktér	vloží jiný nebo opravený soubor, dále se pokračuje od bodu 4 hlavního scénáře

1.5.1.4 UC008 - Uložení layoutu do souboru

Cíl případu užití	Uložení vytvořeného <i>layoutu</i> do souboru		
Vstupní podmínky	Uživatel má vytvořen <i>layout</i>		
Výstupní podmínky	Je vytvořen fyzický soubor s <i>layoutem</i> .		
Základní scénář	Krok	Role	Akce
	1	Aktér	provede volbu pro uložení <i>layoutu</i> .
	2	Systém	Zkontroluje, zda se nepracuje s existujícím souborem. Pokud ano a volba nebyla zadána jako uložit jako, pak se provede UC009 – uložení do stávajícího souboru
	3	Systém	zobrazí dialog pro výběr souboru
	4	Aktér	zadá jméno souboru a potvrdí uložení
	5	Systém	zapiše <i>layout</i> do souboru a zobrazí potvrzení, pokud chyba – UC010 – chyba zápisu
Alternativní scénář UC009	Krok	Role	Akce
Uložení do stávajícího souboru	2a1	Systém	uloží <i>layout</i> do otevřeného souboru a zobrazí potvrzení, pokud chyba – UC010 – chyba zápisu
Alternativní scénář UC010	Krok	Role	Akce
Chyba zápisu	5a1	Systém	zobrazí chybové hlášení, že je do souboru nelze zapisovat
	5a2	Aktér	vybere jiný soubor a poté pokračuje hlavní scénář od bodu 4

1.5.1.5 UC011 - Načtení layoutu ze souboru

Cíl případu užití	Otevření <i>layoutu</i> uloženého v souboru pro další editaci		
Vstupní podmínky	Uživatel má k dispozici databázi typů strojů a uložený <i>layout</i>		
Výstupní podmínky	Uložený <i>layout</i> je načten		
Základní scénář	Krok	Role	Akce
	1	Aktér	Provede volbu pro načtení <i>layoutu</i> ze souboru.
	2	System	Zobrazí dialog pro výběr souboru
	3	Aktér	Vybere soubor k otevření a potvrdí otevření
	4	System	Načte soubor a zobrazí <i>layout</i> v editoru, pokud nastane chyba čtení - UC012 - chyba čtení souboru
Alternativní scénář UC012	Krok	Role	Akce
	4a1	System	zobrazí chybové hlášení - soubor nečitelný
Chyba čtení souboru	4a2	Aktér	odstraní ochranu zápisu a pokračuje hlavní scénář od bodu 3

1.5.1.6 UC013 - Uložení layoutu do databáze

Cíl případu užití	Uložení <i>layoutu</i> do podnikové databáze		
Vstupní podmínky	Uživatel má vytvořený <i>layout</i> k uložení		
Výstupní podmínky	<i>Layout</i> je uložen do databáze		
Základní scénář	Krok	Role	Akce
	1	Aktér	Provede volbu pro uložení <i>layoutu</i> do databáze <i>layoutu</i>
	2	System	Provede validaci <i>layoutu</i> před uložení, pokud není validní - UC014 - chyba v <i>layoutu</i>
	3	System	Zobrazí dialog o uložení <i>layoutu</i>
Alternativní scénář UC014	Krok	Role	Akce
	2a1	System	zobrazí chybové hlášení o chybách v <i>layoutu</i>
Chyba v <i>layoutu</i>	2a2	Aktér	opraví chyby a pokračuje hlavní scénář od bodu 1

1.5.1.7 UC015 - Editace stávajícího layoutu

Cíl případu užití	Otevření <i>layoutu</i> uloženého v databázi pro další editaci		
Vstupní podmínky	Uživatel je připojen na databázi		
Výstupní podmínky	Uložený <i>layout</i> je načten		
Základní scénář	Krok	Role	Akce
	1	Aktér	vybere z databáze <i>layout</i> pro editaci
	2	System	Provede UC016 - nalezení <i>layoutu</i> a <i>layout</i> nahraje do editoru
	3	Aktér	Upraví stávající <i>layout</i> a zvolí uložení <i>layoutu</i>
	4	System	Provede UC013 - Uložení <i>layoutu</i> do databáze

1.5.1.8 UC016 - Nalezení layoutu v databázi

Cíl případu užití	Nalezení <i>layoutu</i> v databázi dle zadaného id		
Vstupní podmínky	Uživatel zadal id <i>layoutu</i>		
Výstupní podmínky	Smazání <i>layoutu</i> z podnikové databáze		
Základní scénář	Krok	Role	Akce
	1	System	Provede nalezení <i>layoutu</i> dle zadaného id
	2	System	Pokud je databáze dostupná a vyskytuje se v ní <i>layout</i> pak vrátí odkaz na <i>layout</i> , pokud chyba - UC017 - chyba čtení databáze
Alternativní scénář UC017 chyba čtení databáze	Krok	Role	Akce
	2a1	System	zobrazí chybové hlášení o chybě při prohledávání databáze

1.5.1.9 UC018 - Odstranění layoutu z databáze

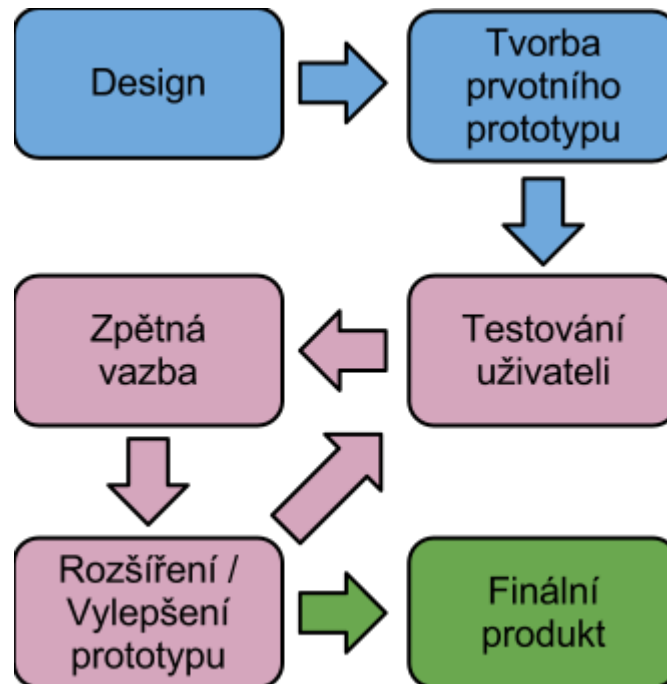
Cíl případu užití	Odstranění <i>layoutu</i> uloženého v databázi		
Vstupní podmínky	Uživatel je připojen k databázi		
Výstupní podmínky	Vybraný <i>layout</i> je odstraněn z databáze		
Základní scénář	Krok	Role	Akce
	1	Aktér	vybere z databáze <i>layout</i> pro odstranění
	2	System	Provede UC016 - nalezení <i>layoutu</i> a zobrazí dialog pro potvrzení odstranění <i>layoutu</i>
	3	Aktér	Potvrdí odstranění
	4	System	Odstraní <i>layout</i> z databáze

1.5.1.10 UC019 - Duplikování layoutu v databázi

Cíl případu užití	Duplikování <i>layoutu</i> uloženého v databázi		
Vstupní podmínky	Uživatel je připojen k databázi		
Výstupní podmínky	Vybraný <i>layout</i> je v databázi duplikován pod jiným <i>id</i>		
Základní scénář	Krok	Role	Akce
	1	Aktér	vybere z databáze <i>layout</i> pro duplikaci
	2	System	Provede UC016 - nalezení <i>layoutu</i>
	3	System	Vytvoří duplicitní <i>layout</i> v databázi

1.6 Vývojová metoda - Prototyping

Pro tuto aplikaci, kde je jedním z hlavních úkolů vytvořit *user-friendly* grafické rozhraní, byla zvolena metoda prototypování [1]. Konkrétně tzv. *Evoluční prototypování*, kdy je zpočátku vytvořen tzv. inicializační prototyp, následně je prototyp otestován uživateli softwaru a na základě zpětné vazby uživatelů je prototyp dále rozvíjen a testován v menších vývojových cyklech (Obrázek 7).



Obrázek 7: Diagram *Evolučního prototypování* [zdroj vlastní]

Výhodou této metodiky je rychlé poskytnutí funkční verze aplikace a právě díky menším evolučním cyklům dochází k dokonalejšímu přizpůsobení „look&feel“ zákazníkovi. Evoluční cykly a neustálé testování budoucími uživateli má také vliv na selekci důležitých vlastností aplikace a naopak potlačení méně důležitých, nebo dokonce nevyužitých funkcionalit, které byly v základní fázi definovány jako důležité.

2 ANALÝZA TECHNOLOGIÍ

Před samotnou implementací je dobré si provést analýzu technologií, které .NET Framework pro vývoj aplikace poskytuje. Konkrétně jsou to technologie pro implementaci GUI, komunikaci s databází a práci s XML. Z těchto technologií budeme následně vybírat ty nejvhodnější pro implementaci aplikace.

2.1 .NET

.NET Framework tvoří vývojovou platformu pro vývoj a běh desktopových, webových, ale i mobilních aplikací a služeb. Obsahuje dvě hlavní komponenty a to *CLR (Common language runtime)* a knihovny tříd. Jelikož je tato platforma úzce spjata s operačním systémem Microsoft Windows, jedná se o velmi rozšířenou a v poslední době oblíbenou platformu.¹

K výhodám vývoje na platformě .NET patří bohatost jejich knihoven, relativní jednoduchost a efektivita vývoje. Díky pomyslnému velmi silnému spojení platformy s komerčním prostředím je k dispozici propracovaná dokumentace a v neposlední řadě může být jako plus považována neustálá aktualizace platformy a s tím související bezpečnost kódu v knihovnách.

.NET Framework má samozřejmě i své nevýhody. K nim jednoznačně patří její nepoužitelnost, respektive omezená použitelnost v prostředí GNU/Linux, kde je pro běh aplikací nutno použít jeden z neoficiálních portů .NET Frameworku pro prostředí GNU/Linux. Další nevýhodou je jeho slabší výkon oproti nativnímu kódu, ten je způsoben zejména dodatečným JIT (Just In Time) překladem [2].

Právě bohatost knihoven frameworku umožňuje vývoj všech hlavních typů aplikací. Od konzolových aplikací přes služby až po webové aplikace. Jako nejdůležitější části knihoven můžeme zmínit tyto [3]:

- ASP.NET – pro vývoj webových aplikací

¹Od verze Windows Vista je .NET Framework 2.0 přímo součástí operačního systému

- WPF(Windows Presentation Foundation) a Windows Forms – pro tvorbu aplikací s GUI
- WCF(Windows Communication Foundation) – vývoj webových služeb a komunikační infrastruktury aplikací
- LINQ(Language Integrated Query) – knihovny pro objektový přístup k datům v databázích, XML a objektech

2.1.1 Technologie pro tvorbu GUI

Pro tvorbu GUI poskytuje .NET Framework od verze 3.0 hned dvě technologie pro tvorbu grafického uživatelského rozhraní.

WinForms

Tou starší z nich a možná i známější jsou WinForms. Jedná se o API zprostředkávající řízený přístup ke knihovnám pro tvorbu uživatelského rozhraní v rámci nativního Windows API. WinForms poskytují řadu předdefinovaných komponent k pohodlnému vývoji grafického rozhraní. Nevýhodou WinForms je hlavně v konceptu vytváření grafické podoby. Celý grafický návrh je vytvářen v rámci zdrojového kódu aplikace, kdy je sice možné oddělit kód GUI a kód v pozadí pomocí použití tzv. partial class (neboli rozdělení definice třídy na několik částí), nicméně i přesto, že řada dostupných vývojářských nástrojů poskytuje grafické nástroje pro tvorbu formulářů, je tvorba GUI složitější, a ve výsledku propojuje práci programátora a designera více, než je tomu u druhé technologie, kterou .NET Framework poskytuje.

Windows Presentation Foundation

Touto druhou technologií je technologie WPF (Windows Presentation Foundation), ta posunuje možnosti podoby GUI o mnoho dále. Hlavním rozdílem je, že veškerá grafika vytvářená pomocí WPF je renderována za pomoci knihoven DirectX a tedy poskytuje možnost jednoduše začlenit do uživatelského rozhraní další graficky náročnější prvky. Mezi tyto prvky patří například 2D a 3D animace, vektorová a rastrová grafika, ale i multimediální obsah (audio a video). Z hlediska implementace GUI používá WPF oddělenou prezentační část a funkční část aplikace. Zatímco prezentační část může být definována ve značkovacím jazyce XAML, funkční část je definována v jednom z mnoha jazyků .NET Frameworku. Přínosem tohoto je zpřehlednění kódu aplikace a v praxi zjednodušení paralelizace práce vývojářů a návrhářů uživatelského rozhraní.

Další výhodou je tzv. *databinding*, který umožňuje snadnou cestou zprostředkovat presentování a interakci s daty prostřednictvím grafických prvků.

2.1.2 Technologie pro přístup k databázi

Základním přístupem pro práci s databází je využití jazyka SQL a připojení k databázi pomocí objektů *SqlConnection* a *SqlCommand*, díky kterým můžeme dané příkazy nad databází spouštět. V příkladu je uveden běžný dotaz typu SELECT a jeho spuštění nad databází.

```
1 void SqlCommandExample(){
2     string queryString = "SELECT * FROM dbo.Items;";
3     using (SqlConnection connection = new SqlConnection(connectionString))
4     {
5         SqlCommand command = new SqlCommand(queryString,connection);
6         connection.Open();
7         SqlDataReader reader = command.ExecuteReader();
8         try
9         {
10            while (reader.Read())
11            {
12                Console.WriteLine(String.Format("{0}, {1}",
13                    reader[0], reader[1]));
14            }
15        }
16        finally
17        {
18            reader.Close();
19        }
20    }
21 }
```

Obrázek 8: Příklad přístupu k databázi pomocí *SqlCommand*

Tento přístup je velice univerzální pro širokou škálu databázových serverů a umožňuje spouštět všechny typy dotazů včetně TCL (*Transaction control language - příkazy pro management transakcí*) a DDL (*Data definition language – příkazy pro definici struktury a schématu databáze*) příkazů.

2.1.2.1 LINQ to SQL

LINQ (Language Integrated Query) je obecně komponenta .NET Frameworku, která umožňuje začlenit dotazy nad daty do programovacích jazyků frameworku bez použití komplikovaných konstrukcí. Komponenta definuje metody pro dotazování, třídění a operace nad daty. *LINQ to SQL* je podмноžina metod umožňující pracovat s daty uloženými na SQL Serveru (konkrétně Microsoft SQL Server, nicméně existují i porty pro MySQL a Oracle). Operace poté probíhají tak, že tzv. *LINQ statement* je přeložen do SQL

příkazu a spuštěn na serveru. Výsledek dotazu je následně převeden do objektů, které byly na danou entitu v databázi mapovány. Obrázek 9 obsahuje ukázkou mapování objektu na tabulku v databázi a následně dotaz na tento objekt. Do objektu třídy *Customer* pak získáme prvního zákazníka s *Id* větším jak tři, a pokud takový neexistuje, je vybrána výchozí možnost.

```
1 [Table(Name="Items")]
2 public class Customer
3 {
4     [Column(Name="Id",IsPrimaryKey = true)]
5     public int ItemKey;
6     [Column(Name="Name")]
7     public string CustName;
8 }
9
10 public void SqlDeserialize(int id){
11     DataContext con = new DataContext(DataSourceClass.DB_CONNECTION_STRING);
12     Customer cust = (from it in con.GetTable<Customer>()
13                     where it.ItemKey > 3
14                     select it).FirstOrDefault();
15 }
```

Obrázek 9: Příklad mapování třídy a dotaz nad databází

2.1.3 Technologie pro přístup k XML

Kromě běžného procházení a vytváření XML dokumentu pomocí tříd *XmlReader* a *XmlWriter* můžeme k datům uloženým v dokumentu XML přistupovat opět pomocí komponenty LINQ.

Konkrétně k tomuto účelu slouží LINQ to XML. S touto „verzí“ komponenty LINQ se pracuje naprosto stejně jako s „verzí“ pro SQL. Tvorba i rozsáhlé XML struktury se díky komponentě LINQ velice zjednoduší. Jako příklad uvádím vytvoření jednoduché XML struktury.

```
1 XDocument doc = new XDocument(
2     new XDeclaration("1.0", "utf-8", "yes"),
3     new XElement("SimpleElement",
4         new XAttribute("attribute1", "123"),
5         new XAttribute("attribute2", "Brooklyn, NY")));
```

Obrázek 10: Ukázka vytvoření XML struktury

```
1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <SimpleElement attribute1="123" attribute2="Brooklyn, NY">
3 </SimpleElement>
```

Obrázek 11: Výsledná struktura XML dokumentu

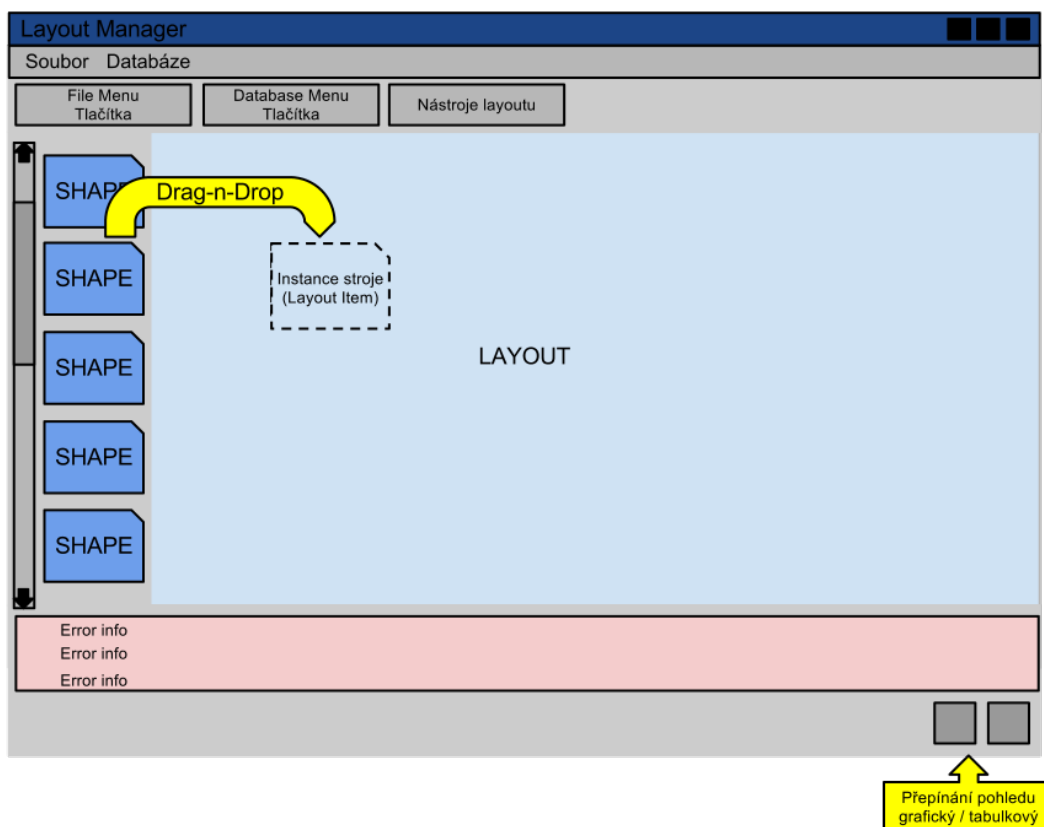
II. PRAKTICKÁ ČÁST

3 DESIGN

V předchozí části tedy byl proveden rozbor požadovaných vlastností a funkcionalit aplikace. Z tohoto rozboru je nutné načrtnout podobu aplikace, a to jak její grafickou podobu a ovládání, tak datové struktury.

3.1 Návrh GUI

Podle analýzy požadavků má aplikace obsahovat jednoduše a intuitivně ovladatelné GUI. Výsledkem této části návrhu bude tzv. *wireframe* [4], jakýsi prototyp grafického rozhraní, nad kterým je možné později se zadavatelem diskutovat a před samotnou realizací měnit jeho podobu.

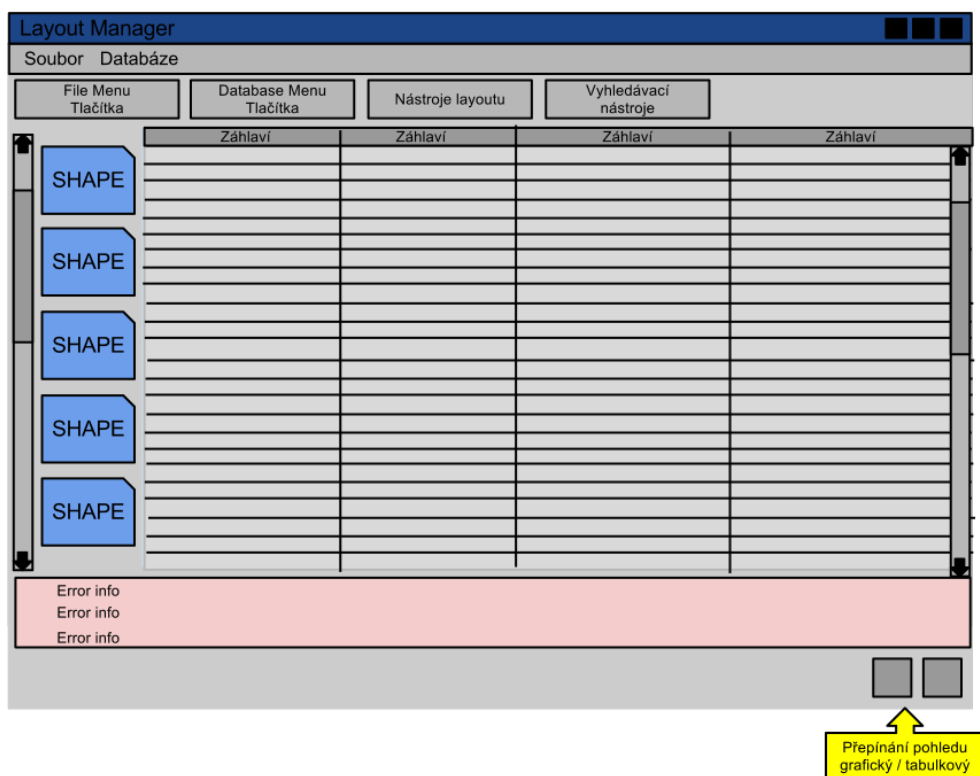


Obrázek 12: Návrh grafického zobrazení

Návrhu GUI byl inspirován grafickým rozhraním některých CAD systému (*Autodesk – AutoCAD, Inventor*), kde je možné do výkresu pomocí metody Drag&Drop vkládat předem definované bloky (v tomto případě *shape*). Pro seznam bloků, který se bude nacházet v levé části GUI, byla upřednostněna grafická podoba *shape* před jejich textovou reprezentací, což usnadňuje a urychluje orientaci v seznamu *shape*. Tento panel bude možné skrýt tak, aby po přidání všech potřebných strojů nepřekážel při jejich případném

přemísťování a editaci. Editace vlastností strojů bude přístupná v grafickém zobrazení vyvoláním dialogu z kontextového menu nebo pouhým dvojklikem na upravovaný stroj. Přesouvání strojů bude intuitivní tažením kurzoru po ploše *layoutu*.

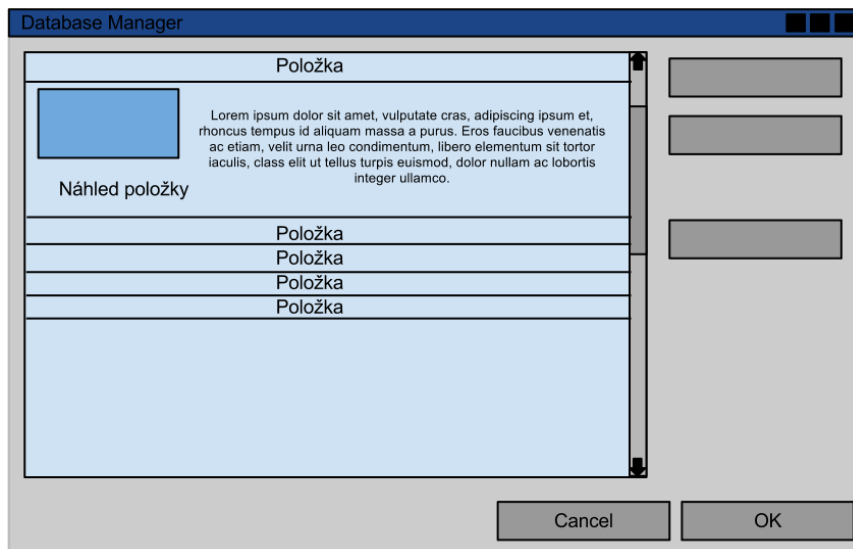
Přepnutí do tabulkového rozhraní bude realizováno tlačítky ve stavové liště. V tomto zobrazení bude možné editovat všechny atributy strojů přímo v tabulce. Odebírání a kopírování bude dostupné přes kontextové menu nebo pomocí klávesových zkratk. Při zobrazení tabulkového rozhraní se v nástrojové liště zobrazí možnosti vyhledávání v tabulce strojů, to by mělo umožnit rychlejší editaci a orientaci v položkách *layoutu*.



Obrázek 13: Návrh tabulkového rozhraní

V dolní části hlavního okna bude tabulka zobrazující chyby v *layoutu*, tak jako tomu je v některých vývojových prostředích. Při kliknutí na konkrétní chybu se chyba v *layoutu* zvýrazní, popřípadě se vyvolá nabídka nebo dialog pro její opravu (konkrétně bude řešeno v implementaci).

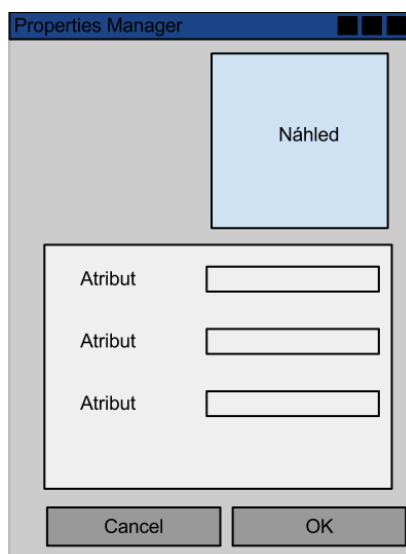
Dialogy pro správu databáze, ukládání a otevření *layoutu* budou otvírány standardně z hlavního menu a zároveň budou pro pohodlí uživatele dostupné v tlačítkové liště.



Obrázek 14: Návrh podoby dialogu pro práci s databází

Obrázek 14 je návrh podoby nástroje pro procházení a náhled do databáze. Podoba nástroje bude v případě správy *layoutů* a *shape* v databázi stejná. V pravé části rozhraní se budou nacházet tlačítka pro operace Otevři/Edituj/Smaž, popřípadě další dle potřeby. V levé části se okna se nachází tabulka záznamů v databázi. Po označení položky v tabulce by se měl pod položkou rozvinout náhled záznamu v databázi včetně grafické reprezentace, to by mělo opět usnadnit orientaci při výběru *layoutu/shape*.

Posledním prvkem GUI je okno pro editaci vlastností strojů, *layoutu* a záznamu *shape* v databázi.



Obrázek 15: Návrh dialogu pro editaci atributů objektu

Okno obsahuje náhled grafické reprezentace objektu pro jakousi rychlou vizuální kontrolu vybraného prvku (může přispět k rozpoznání nechtěně vybraného prvku pro editaci). Dolní část formuláře obsahuje tabulku atributů pro jejich editaci a úplně vespod pak klasická potvrzovací tlačítka.

Po tomto grafickém návrhu rozhraní je vhodné „simulovat“ všechny případy užití a ověřit si, jestli je možné pomocí navrženého GUI docílit všech požadovaných operací. Zároveň tím ověříme i úroveň ergonomie GUI.

3.2 Návrh datových struktur

Návrh datových struktur musí být koncipován tak, aby byl použitelný jak pro třídy objektů využívaných v aplikaci, tak pro realizaci databázové struktury. Provedl jsem tedy napřed rozpracování požadavků, co by měly struktury uchovávat, a dále navrhnul relační model databáze s integritními omezeními.

3.2.1 Požadavky na atributy jednotlivých objektů

Pro návrh jednotlivých atributů byl postup určování atributů následující:

- atributy vyplývající z analýzy funkčních požadavků,
- atributy pro grafickou reprezentaci,
- atributy s dodatečnými informacemi.

Layout

Z analýzy požadavků jsou pro *layout* podstatné informace o uživateli, který *layout* vložil a datum jeho vložení nebo editace. Pro tyto dvě informace je zaveden textový atribut *InsUser* a časový atribut *InsDate*. Dalším požadavkem bylo rozlišení *layoutu* na online *layout* a LCD *layout*. Tuto informaci ponese příznakový atribut *IsWebLayout*. Platnost *layoutu* je uvedena v časových attributech *DateFrom* (platnost od) a *DateTo* (platnost do). Posledním atributem, který vyplývá z předešlé analýzy, je jméno *layoutu* a tedy textový atribut *Name*.

Pro grafickou reprezentaci jsou zavedeny celočíselné atributy *OnlineWidth* a *OnlineHeight* reprezentující rozměry podkladu a speciální atribut *Layout*, který reprezentuje binární podobu obrazových dat.

Pro případný popis *layoutu* nebo dodatečné poznámky k *layoutu* je navrhnut textový atribut *Description*.

LayoutItem

Datová struktura *LayoutItem* reprezentuje instanci stroje v *layoutu*. Z charakteru aplikace vyplývá, že propojení *LayoutItem* a reálného stroje musí být vytvořeno přes unikátní identifikátor, to je v tomto případě atribut *WRKCTRID*. Atribut zajišťuje další komunikaci napříč databází podnikového IS a tedy i získávání dat o stavu stroje a probíhající výroby na něm. *WRKCTRID* je také jediný „funkční“ atribut vyplývající z analýzy požadavků.

Atributy pro grafickou reprezentaci jsou celočíselné atributy *Online_X*, *Online_Y* a *Monitor_Scale*. První dva atributy určují pozici *LayoutItem* v *layoutu* (konkrétněji jeho levý horní roh vůči levému hornímu rohu *layoutu*). Atribut *Monitor_Scale* určuje proporční měřítko grafiky vůči rozměrům výchozího *shape* uloženého v databázi.

Doplňkovým atributem pro určení měřítko je atribut *LCD_Scale*, který slouží jako pomocný atribut pro zobrazení *layoutu* na LCD.

Shape

Struktura *Shape* reprezentuje typ stroje. V případě této datové struktury je situace podobná jako u *layoutu*.

Pro identifikaci uživatele a data vložení jsou zavedeny atributy *InsUser* a *InsDate*. Platnost záznamu je dána atributy *DateFrom* a *DateTo*. Každý tvar je pojmenován a jeho název je uložen v atributu *Name*.

Jako jeden z hlavních atributů je atribut *Shape*, který uchovává obrazová data podoby stroje v binární podobě.

Jako doplňkový atribut umožňující uchování popisu nebo poznámek k *Shape* je zde zaveden atribut *Description*.

3.2.2 Relační model a tvorba databázové struktury

Pro realizaci struktury databáze je potřeba u jednotlivých datových struktur zavést atributy pro jejich identifikaci a vztahy mezi nimi. Součástí tvorby databáze bylo určit integritní omezení.

Po teoretickém návrhu objektů se jeví jako vhodné vytvořit relační model databáze. Zajímavostí je ve struktuře/databázi užití datového typu *IMAGE*, který je v tomto případě využit na uložení obrázkového souboru u struktur *Layout* a *Shape*. Tento datový typ

umožňuje uložení až 2GB souboru přímo do databáze [5]. Vyvarujeme se tak skladování souborů obrázků v adresářích na disku a umožníme tak jejich jednodušší správu

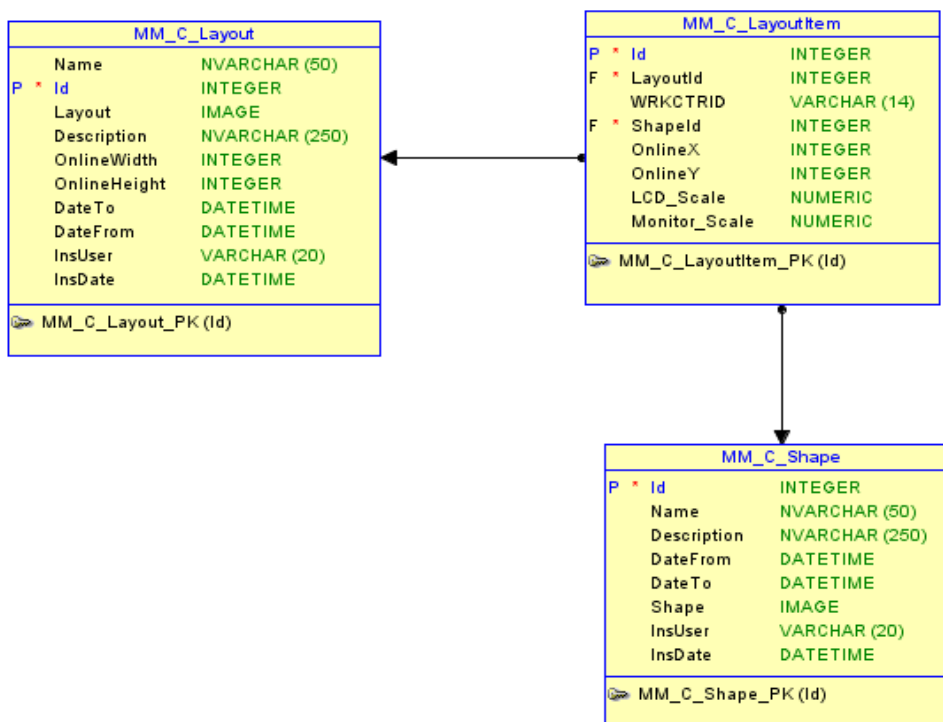
Layout a Shape

U obou struktur je nutné přidat pouze jednoznačný identifikátor *Id*. Ten bude následně sloužit jako reference cizího klíče.

LayoutItem

Pro propojení *LayoutItem* s *layoutem* a *shape* byly přidány cizí klíče *LayoutId* a *ShapeId*. Konkrétně *LayoutId* určuje příslušnost k patřičnému *layoutu*, zatímco *ShapeId* určuje grafickou podobu *LayoutItem*, tedy odkazuje na příslušný typ v tabulce typů strojů.

Pro jednoznačnou identifikaci *LayoutItem* v databázi je atribut *Id* (jelikož *WRKCTRID* není unikátní napříč databázi a může se krýt například v případě existence dvou *layoutů*, jednoho pro LCD a druhého pro online vizualizaci).



Obrázek 16: Relační model databáze

3.2.3 IO integritní omezení

Pro ošetření vstupu (Integritní omezení IO) bylo nutné zvolit přiměřenou hranici nastavení integritních omezení deklarativních, tedy na straně serveru a procedurálních na straně aplikace.

Deklarativně by měla být nastavena opravdu jen nejnужnější omezení, hlavně z důvodu zatížení serveru a rychlosti reakce na porušení integrity. Všechna ostatní integritní omezení budou implementována procedurálně v aplikaci.

Layout a Shape

U tabulky *layoutů* byl nastaven jako NOT NULL atribut *Id*, což je samozřejmostí, jelikož je určen jako primární klíč. Další atribut, jehož hodnota nesmí být NULL, je atribut *Name*. Tento atribut bude zobrazován v nabídkách a při prohlížení databáze, nesmí tedy být v žádném případě prázdný.

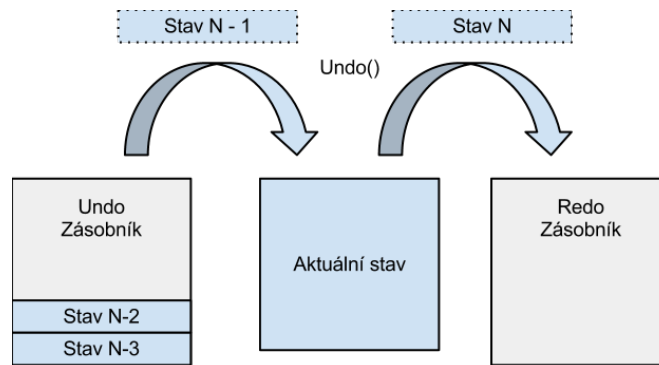
V aplikaci před uložením do databáze bude kontrolována časová souslednost atributů *DateFrom* a *DateTo*. Jako poslední kontrolovaný je atribut *Layout* (resp. *Shape*). Pokud by tato položka byla prázdná, nemá smysl objekt ukládat do databáze.

LayoutItem

V případě *LayoutItem* je jako nenulový nastaven atribut *Id*, dále pak atribut *WRKCTRID*, aby bylo zajištěno propojení *LayoutItem* s reálným zařízením. Jak již bylo zmíněno, atribut *LayoutId* je cizí klíč odkazující na příslušný *layout*. Jelikož by nemělo smysl v databázi uchovávat „instanci stroje“, která by nebyla nikde zobrazena, musí být atribut *LayoutId* nenulový.

3.3 Návrh funkcionality *Undo/Redo*

Ve své podstatě je návrh samotné funkce velice jednoduchý. Jedná se o soustavu dvou limitovaných zásobníků, které obsahují plnou kopii *layoutu* a všech položek v *layoutu*. Limitování zásobníku je nutné ke snížení paměťové náročnosti aplikace.

Obrázek 17: Diagram návrhu funkcionality *Undo*

Po každé provedené akci měnící stav *layoutu* nebo jeho položek je předchozí stav uložen na zásobník pro funkci *Undo*. Pokud je funkce spuštěna, z vrcholu zásobníku se odebere poslední stav, „nahraje se“ do aktuálního stavu a zároveň se aktualizuje zásobník pro funkci *Redo*, kde je uložen aktuální stav. Funkce *Redo* je totožná jen s prohozenými zásobníky.

4 PRŮBĚH VÝVOJE

Jelikož popisovat implementaci jednotlivých částí v průběhu vývojových cyklů by bylo zdouhavé a ve výsledku by vznikl nepřehledný popis aplikace, rozhodl jsem se stručně popsat jednotlivé vývojové cykly, kterými aplikace prošla, a vysvětlit okolnosti některých nestandardních řešení. Konkrétní implementaci popsat až v sekci 5, kde bude popsána finální verze aplikace.

4.1 Volba technologií

Pro kód aplikace je zadavatelem vyžadován programovací jazyk C#, proto tedy není nutné zabývat se výběrem programovacího jazyka.

Pro realizaci GUI byla zvolena technologie WPF, jejíž výhody jsou popsány v sekci (2.1.1).

Pro komunikaci s databází a práci s XML soubory využijí LINQ komponenty .NET Frameworku. Díky tomu, že databáze bude běžet na Microsoft SQL Serveru, můžeme komponentu *LINQ to SQL* bez nejmenších potíží použít. Toto nám z velké části usnadní práci s databází.

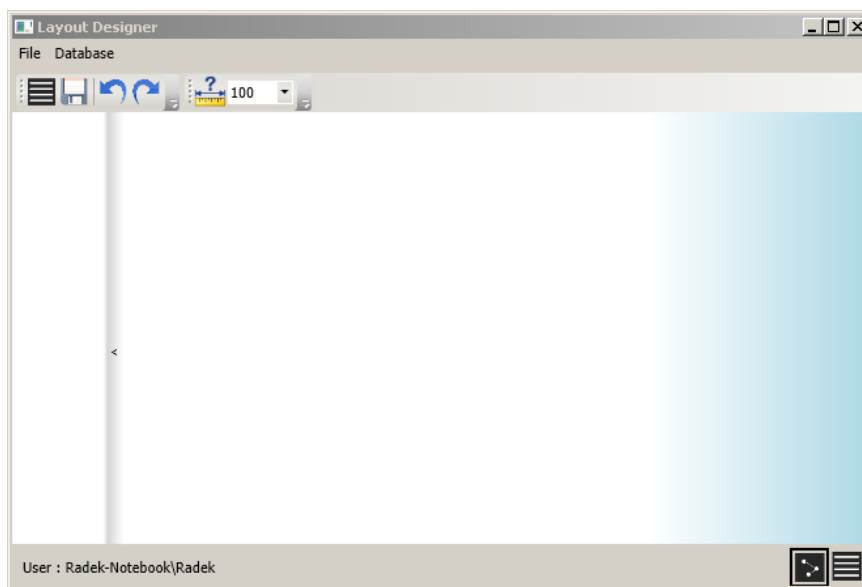
4.2 Inicializační prototyp

Implementace inicializačního prototypu je jedna z nejnáročnějších fází. Zahrnuje implementaci všech navrhovaných prvků a funkcionalit, které budou v následujících „evolučních“ fázích testovány a rozšiřovány dle zpětné vazby testerů.

Jelikož byly paralelně s Layout Designerem vyvíjeny další „závislé komponenty“ pro zobrazování *layoutů*, byla již po návrhu datových objektů vytvořena databázová struktura. Při prvotním testování bylo tedy nutné upravit drobné rozdíly v databázových atributech tak, aby se dosáhlo kompromisu mezi lokální databází, která byla použita pro vývoj inicializačního prototypu, a „živou“ podnikovou databází. Zároveň tyto změny uzpůsobit tak, aby byl již vytvořený inicializačním prototyp ovlivněn co nejméně.

V této fázi inicializační prototyp obsahoval všechny hlavní funkcionality pro tvorbu *layoutu*, implementaci všech datových objektů a hlavních ovládacích prvků. Protože se jednalo o prototyp, aplikace neobsahovala klávesové zkratky a funkcionality poskytující *user-friendly* rozhraní, včetně kontroly chyb.

V rámci inicializačního prototypu bylo naimplementováno jen jednoduché rozhraní pro práci s databází. To obsahovalo nástroj pro správu tvarů strojů umožňující přidávání tvarů strojů do databáze a dále nástroj umožňující načtení a uložení *layoutu* do databáze.



Obrázek 18: Ukázka inicializačního prototypu

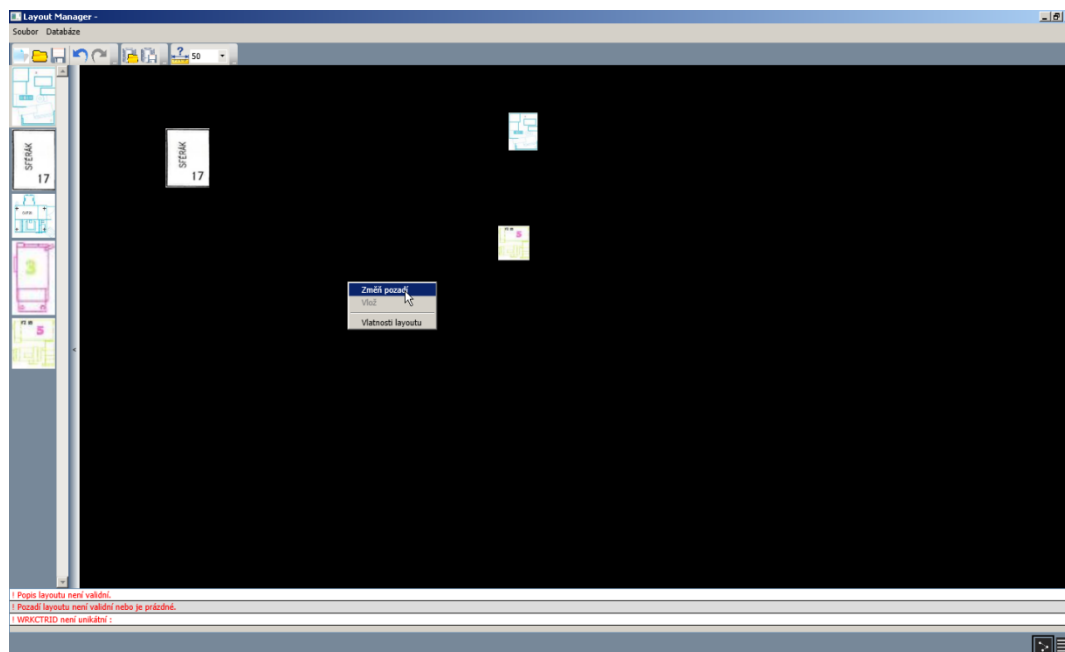
Součástí inicializačního prototypu byly implementovány základní funkcionality pro práci se soubory. (Tedy funkce *Save* a *Open*).

Po otestování prototypu byly upraveny některé funkce týkající se práce s databází a dále navržena některá vylepšení grafického rozhraní. Ta se týkala hlavně úprav zobrazení některých tabulek. Testování odhalilo i některé nedostatky ve funkcionalitě *Undo* a *Redo*, zejména již nepotřebný seznam předchozích stavů při založení (načtení) nového *layoutu*.

4.3 První evoluční cyklus

Do prvního evolučního cyklu byly zahrnuty úpravy vyplývající z testování inicializačního prototypu a dále byla aplikace rozšiřována o prvky usnadňující práci s aplikací.

Došlo k rozšíření správy databáze o editaci položek v databázi a v případě správy *layoutů* o jejich možnost vymazání a duplikace *layoutu* v databázi. Jednotlivé nástroje procházení databáze byly rozšířeny o detailní náhled vybraného prvku.



Obrázek 19: Ukázka prototypu v prvním evolučním cyklu

Uživatelské rozhraní (Obrázek 19) bylo rozšířeno o klávesové zkratky. Ty umožňují rychlý přístup k úpravám *layoutu* (kopírování, vyjmutí objektů). Dále byl implementován skupinový výběr objektů v *layoutu*.

Z plánovaných funkcionalit byla implementována kontrola chyb v *layoutu* a interakce uživatele s tabulkou chyb.

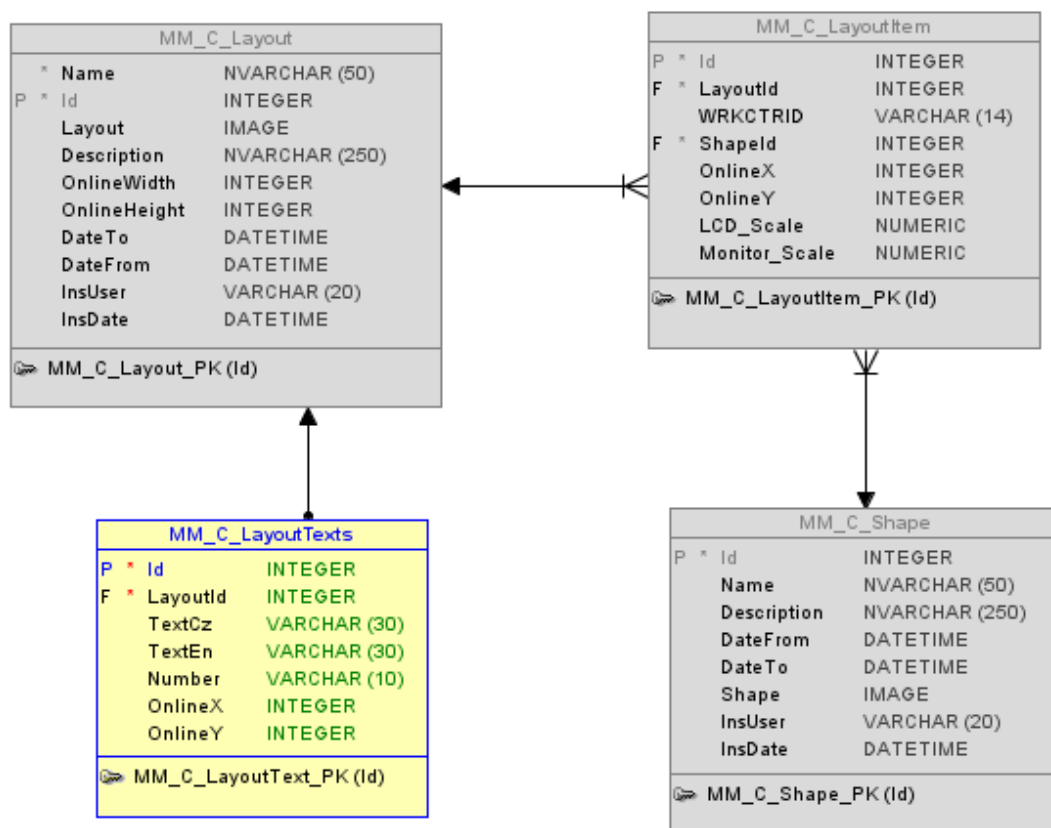
Prototyp byl opět otestován, bylo odhaleno několik chyb týkajících se hlavně grafické úpravy podoby *layoutu*. Hlavně tedy chybné chování při výběru prvků a jejich posunu. K dalším odhaleným chybám patřily chyby při duplikování *layoutu* v databázi, kdy se při vytvoření duplikátu nevytvářely duplicitní záznamy pro objekty v *layoutu*.

4.4 Druhý evoluční cyklus

Během druhého evolučního cyklu aplikace došlo k poměrně velkému rozšíření, které zahrnovalo přidání dalšího objektu do datových struktur. A to prvku *LayoutText*, který má reprezentovat doplňující popisky v rámci *layoutu*. Tato úprava vznikla na popud oddělení zabývajícího se právě projektem „Machine monitoring“.

Zde se projevil problém, který jsem zmiňoval v souvislosti s komplexní analýzou požadavků. Čím později jsou prováděny náročnější komplexnější změny v aplikaci, tím složitější je jejich implementace.

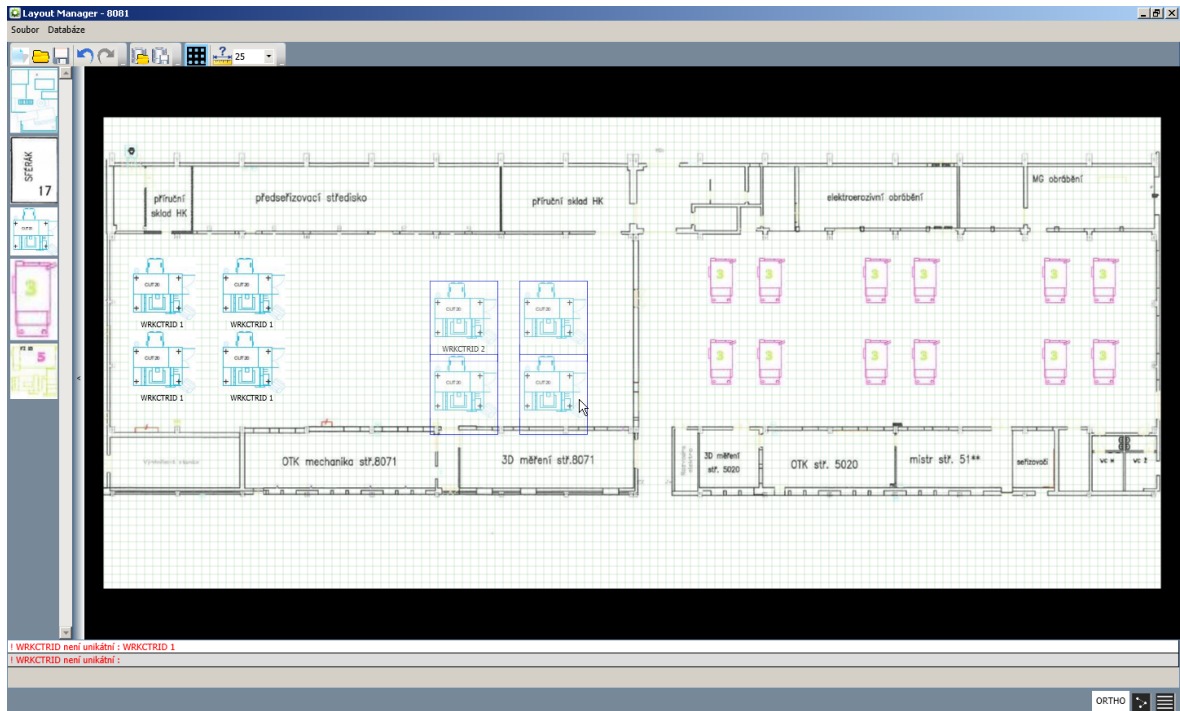
Bohužel základní návrh aplikace s takovýmto rozšířením nepočítal a v tomto stádiu aplikace jsem se rozhodl vytvořit třídu *LayoutText* velice podobnou třídě *LayoutItem*, která obsahuje téměř duplicitní kód. Ve spojitosti s novou třídou přítomnou v *layoutu* byla vytvořena i v databázové struktuře tabulka *LayoutTexts* (Obrázek 20) obsahující všechny potřebné atributy pro uložení popisku do databáze. Naštěstí všechny tyto úpravy nevyžadovaly zásadní změny v algoritmech *GUI*, a tedy implementace těchto úprav nebyla až tak časově náročná. Nicméně tato změna měla negativní dopad na čistotu kódu, ale i při takovéto rozsáhlé změně byla zachována jeho přehlednost.



Obrázek 20: Struktura databáze po rozšíření o *LayoutTexts*

Pokud by tato úprava byla zahrnuta již při návrhu aplikace, řešením by bylo navržení základní třídy, od které by byly dané třídy *LayoutItem* a *LayoutText* odvozeny.

Po tomto rozšíření a implementaci dalších *user-friendly* funkcí jako jsou funkce *ORTHO*, *SnapToGrid* a dalších užitečných klávesových zkratk došlo opět k testování aplikace.



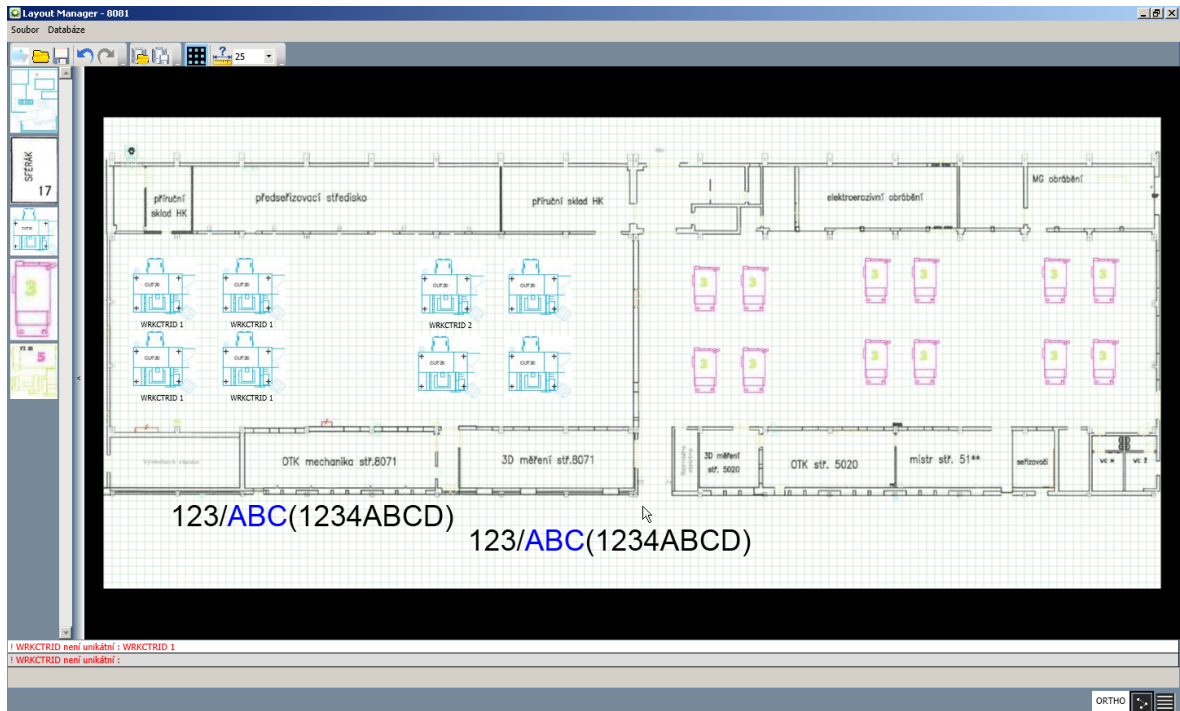
Obrázek 21: Prototyp v druhém evolučním cyklu (funkce *ORTHO* a *SnapToGrid*)

V této fázi testování byla ověřována funkčnost rozšíření o třídu *LayoutText* a zároveň použitelnost aplikace. V souvislosti s tímto testováním byly opět navrženy některé změny ulehčující a urychlující práci s layoutem. Příkladem může být otevření dialogu editace objektu v layoutu dvojklikem a okamžité zaměření kurzoru na textové pole s nejvýznamnějším prvkem. Pro uložení provedených změn a zavření editačního dialogu je nyní možné využít mimo tlačítka *OK* také stisk klávesy *Enter*.

Tyto a podobné úpravy chování zpříjemňují práci s celým uživatelským rozhraním.

4.5 Třetí evoluční cyklus

Třetí evoluční cyklus v podstatě uzavírá evoluční část aplikace a zahrnuje odladování všech implementovaných *user-friendly* funkcí a vlastností uživatelského rozhraní. Dále je v tomto cyklu dotvářena podoba *GUI* z hlediska celkového vzhledu aplikace. V celé aplikaci je realizována lokalizace pro české a anglické prostředí.



Obrázek 22: Prototyp v třetí evoluční fázi

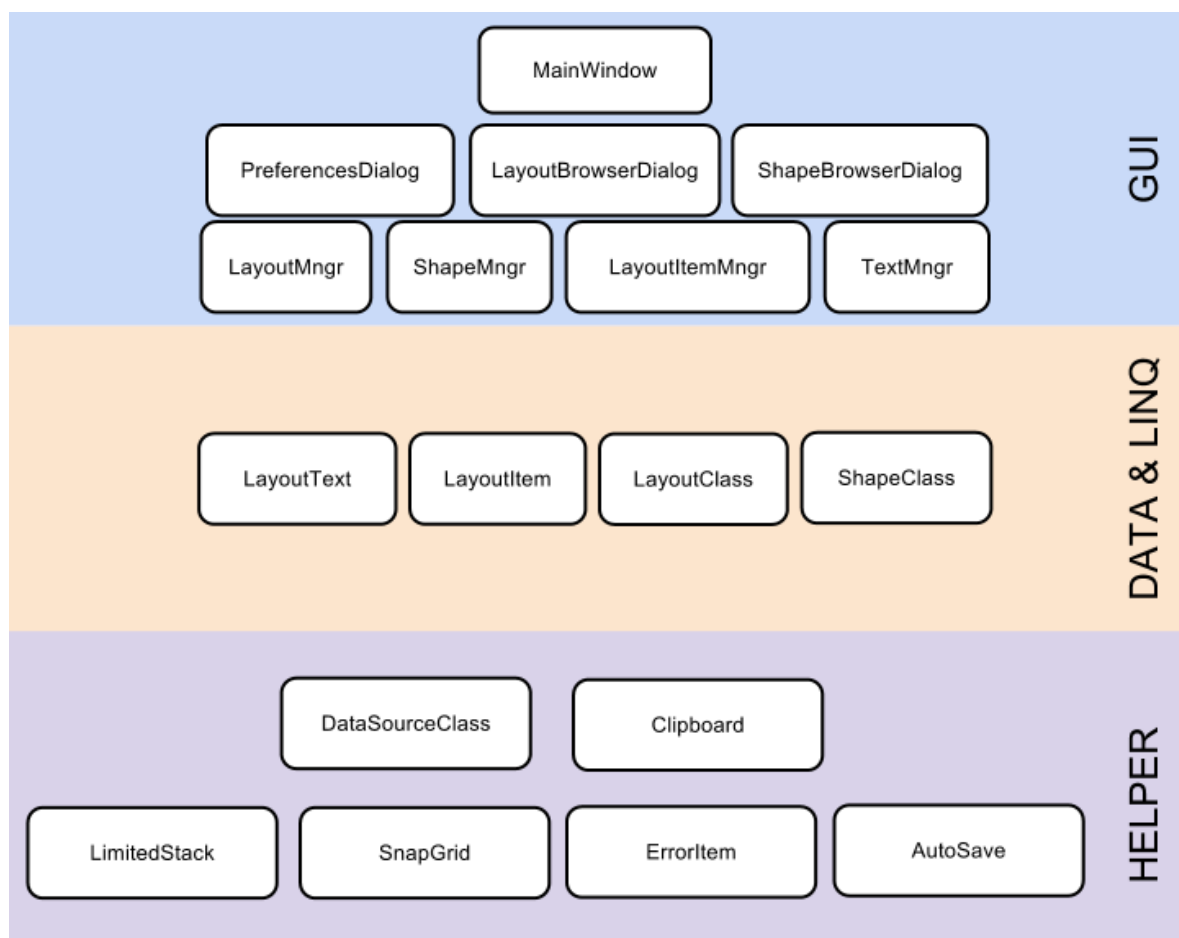
V rámci testování v třetím evolučním cyklu byly vytvářeny zkušební *layouts* pro související aplikace, částečně se tak testovalo reálné nasazení v systému. Na základě tohoto testu byly navrženy další drobné úpravy vzhledu aplikace a jejího chování, tyto úpravy už vedly k vytvoření finálního produktu. Ucelený popis finální verze aplikace bude uveden v následující kapitole.

5 FINÁLNÍ PRODUKT

V této části bude uveden ucelený popis konečné verze aplikace, která bude nasazena v reálném prostředí. Budou zde popsány třídy aplikace a jejich hlavní metody a implementované funkcionality.

5.1 Popis aplikace

Aplikace obsahuje dvě hlavní části a třetí část s pomocnými funkcemi. První část označená v blokovém schématu (Obrázek 23) jako GUI je implementace grafického rozhraní hlavního okna, dialogů a jejich obslužných rutin. Třída hlavního okna spouští veškeré operace na pozadí, udržuje v paměti seznam objektů momentálně se vyskytujících v *layoutu* a aktuálně aktivní *layout*. Tato část je nejobsáhlejší z celého projektu, protože zajišťuje veškerou interakci s uživatelem, ošetřuje nestandardní chování uživatele a zajišťuje všechny funkce vytvářející *user-friendly* prostředí.



Obrázek 23: Blokové schéma tříd aplikace Layout Designer

Druhá část aplikace označená ve schématu (Obrázek 23) jako *DATA & LINQ*, která by se dala nazvat datová, definuje třídy pro propojení objektů zobrazovaných v ploše *layoutu* a právě mapování jednotlivých atributů na atributy databázových objektů (využívající *LINQ*). V této pomyslné druhé části aplikace jsou též implementovány rutiny svazující grafickou a datovou reprezentaci objektů.

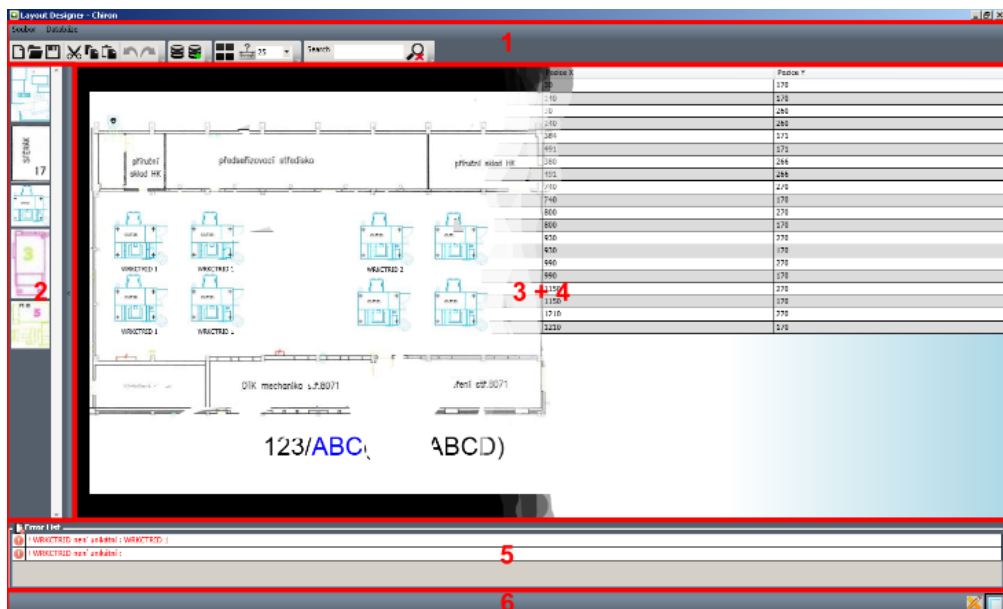
Třetí část obsahuje rutiny pro management nastavení aplikace, převody datových typů, načítání dat do paměti a další pomocné funkce.

5.2 Třídy GUI

Začneme tedy popisem tříd implementujících uživatelské rozhraní. Při popisu tříd budou uváděny jen ty nejdůležitější funkcionality a méně obvyklá nebo zajímavá řešení.

5.2.1 MainWindow

Třída *MainWindow* je stěžejním prvkem celé aplikace. Je to třída hlavního okna aplikace, základ uživatelského rozhraní a interakce s uživatelem. Grafická podoba okna je definována v souboru *MainWindow.xaml* a obsahuje šest hlavních prvků pro ovládání aplikace (Obrázek 24 a Tabulka 2). Tyto prvky jsou grafické komponenty ze jmenného prostoru *System.Windows.Controls*:



Obrázek 24: Rozložení prvků hlavního okna

	Jméno	Třída	Funkce
1		Menu (ToolBarTray)	Hlavní menu a toolbar
2	MainListBox	ListBox	Nabídka <i>shape</i>
3	MainScrollViewer (MainCanvas)	ScrollViewer (Canvas)	Grafická úprava <i>layoutu</i>
4	RowView	DataGrid	Tabulka prvků v <i>layoutu</i>
5	ErrorListBox	DataGrid	Tabulka chyb v <i>layoutu</i>
6		StatusBar	

Tabulka 2: Tabulka hlavních komponent třídy *MainWindow*

5.2.1.1 Načtení *Shape* do nabídky

Načtení nabídky *shape* do *MainListBoxu* je implementováno v obsluze události inicializace okna (*private void OnWindowInitialized(object sender, EventArgs e)*). V této metodě dojde k načtení inicializačního souboru, který je umístěn v systémovém adresáři pro data aplikací *AppData* (*Data Aplikací*) v podadresáři *Layout Designer* pod názvem *Config.xml*. V tomto souboru jsou uvedeny detaily pro připojení k databázi. Při prvním spuštění není tento soubor (složka) k dispozici, proto je v tomto případě vyvoláno dialogové okno pro nastavení připojení k databázi. Při zamítnutí je tento dialog možné vyvolat z hlavního menu *GUI*. Při úspěšném připojení k databázi dojde k okamžitému zálohování nabídky *shape* do souboru *ShapeList.xml*, který je ve stejném adresáři jako konfigurační soubor. Soubor ve formátu XML obsahuje úplný seznam *shape* z databáze včetně obrazových dat a slouží jako knihovna *shape* pro práci v offline režimu (struktura viz Obrázek 25).

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <List>
3   <Shape Id="1"
4     Name="Shape 0" Data="" Description=""
5     DateFrom="2001-01-12T00:00:00" DateTo="2012-12-09T00:00:00"
6     InsUser=""
7     InsDate="2012-03-12T21:23:29.1138935+01:00" />
8 </List>

```

Obrázek 25: XML struktura pro uložení *shape* v souboru *ShapeList.xml*

Struktura *shape* je totožná s navrhovanou datovou strukturou v části (3.2.2), zajímavostí je zde uložení obrazových dat (atribut *Data* elementu *Shape*). Ta jsou uložena v tomto atributu v Base64 kódování, což je kódování umožňující uložení binárních dat do znakového řetězce.

Při zkoumání vlivu tohoto zálohování na rychlost startu aplikace jsem došel k závěru, že tato funkcionality nebude nějak markantně zpomalovat start aplikace. Po konzultaci se zadavatelem jsme navíc došli k závěru, že samotná nabídka bude obsahovat řádově desítky strojů a tedy zpomalení startu aplikace bude přijatelné.

V běžící aplikaci jsou *shape* uloženy v seznamu *MlstImageLibrary*, který je také zdrojem pro *MainListBox*, pomocí něhož je zobrazována paleta dostupných strojů v hlavním okně. Způsob zobrazení jednotlivých *shape* je nadefinován v XAML souboru (Obrázek 26).

```
1 <ListBox.ItemTemplate>
2   <DataTemplate>
3     <Image Source="{Binding Path=ShapeBitmap}"
4       Width="60"
5       MouseMove="MainListBoxMouseMove"
6       Margin="2"/>
7   </DataTemplate>
8 </ListBox.ItemTemplate>
```

Obrázek 26: Definice zobrazení *shape* v *MainListBoxu*

Zde je vidět, že zobrazovaný „náhled“ v *MainListBoxu* je nabindován na property *ShapeBitmap* typu *BitmapImage*.

5.2.1.2 Implementace grafické editace layoutu

Většina interakce je skryta právě za elementem *MainCanvas*, který je obalen elementem *ScrollViewer* umožňujícím scrollování. Element *MainCanvas* má naimplementováno kontextové menu obsahující nabídku pro změnu podkladu *layoutu* a zároveň umožňuje vyvolat dialog pro změnu ostatních atributů *layoutu*. Po změně podkladu *layoutu* dojde i ke změně rozměrů prvku *MainCanvas*, který v tomto případě převezme rozměry podkladového obrázku a tyto rozměry se stávají i aktuálními rozměry *layoutu*. Pokud je rozměr *layoutu* větší než zobrazovaná plocha v *ScrollViewer*, je automaticky umožněno scrollování *layoutu*. Aktuální podoba a vlastnosti *layoutu* jsou udržovány v instanci třídy *LayoutClass* s názvem *Layout*. Ta reprezentuje aktuální stav a jsou s ní nadále prováděny operace související s aktualizací stavu, vracení změn a dalších, které si popíšeme níže.

Přidávání *LayoutItem* do layoutu

Jednotlivé *LayoutItem* se do *layoutu* (v našem případě *MainCanvas*) přidávají pomocí Drag&Drop z *MainListBoxu*. Inicializace Drag&Drop je implementována v event handleru *MainListBoxMouseMove*. V této obslužné rutině je volána metoda

System.Windows.DragDrop.DoDragDrop, v níž je jako parametr *data* nastaven *shape*, který je vybrán v *MainListBoxu*.

Po upuštění (akci „drop“) je vytvořen nový objekt typu *LayoutItem* s patřičně přiřazeným *ShapeId* a aktuální pozicí na souřadnicích určených pozicí kurzoru vzhledem k prvku *MainCanvas*.

Pozice *LayoutItem* v *layoutu* je dána nastavením hodnot atributu *Margin* pro *UserControl* nastavením hodnot „top“ a „left“, který určuje pozici levého horního rohu *LayoutItem*, a to vzhledem k levému hornímu okraji prvku *MainCanvas* (potažmo *layoutu*). Další pohyb *LayoutItem* na ploše *layoutu* (prvku *MainCanvas*) je implementován v event handlerech *MainCanvasMouseMove*, *MainCanvasMouseLeftButtonDown* a *MainCanvasMouseLeftButtonUp*, ty zajišťují výběr *LayoutItem* (ten bude rozebrán níže) a manipulaci s nimi tažením kurzoru po *layoutu*.

Všechny *LayoutItem* jsou udržovány v seznamu *MlstDeviceList* v třídě okna *MainWindow*. Všechny *LayoutItem* bychom mohli udržovat v kolekci *Children* (*System.Windows.Controls.UIElementCollection*) prvku *MainCanvas*, v tomto případě by ale kolekce *Children* obsahovala i elementy jiného typu a museli bychom v kolekci prvky vybírat. To by zvýšilo výpočetní náročnost operací. Proto byl zaveden samostatný seznam *UIElementů* (jejíž podmnožinou je *UserControl* a tedy i *LayoutItem*), nad kterým mohou být prováděny různé operace (např. hromadná změna atributů, selekce, hromadné rozšiřování), aniž by ovlivňovaly grafické rozhraní. Po provedení operace je zavolána funkce *MainCanvasRefresh*.

Funkce *MainCanvasRefresh* vykresluje podobu *layoutu*, mimo vykreslování jednotlivých *LayoutItem* do plochy *layoutu* vykresluje také pozadí *layoutu*, a pokud je potřeba, tak mřížku pro funkci *SnapToGrid* (implementaci *SnapToGrid* bude popsána v sekci (5.2.1.6)). Funkce *MainCanvasRefresh* je spouštěna po jakékoliv operaci související se změnou grafické podoby *layoutu*.

Úprava atributů již existujících *LayoutItem* probíhá pomocí dialogového okna vyvolaného buď dvojklikem na upravovaný objekt nebo z kontextového menu *LayoutItem*.

Přidávání *LayoutText* do layoutu

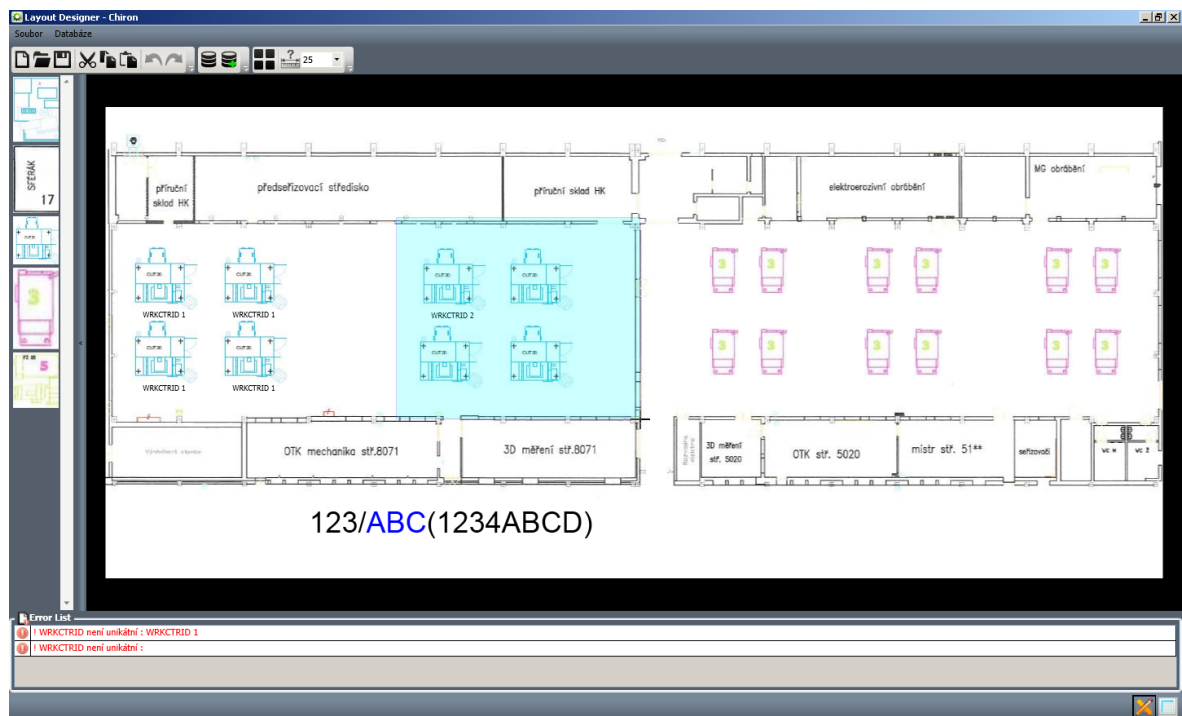
Vložení popisku do layoutu je implementováno pomocí kontextového menu prvku *MainCanvas*. Kdy pomocí volby „Vlož text“ je vyvolán dialog pro zadání obsahu popisku a na místě, kde byla vyvolána kontextová nabídka, je vytvořen patřičný *LayoutText*.

Upravovat pozici popisku je možné stejně jako u *LayoutItem* pomocí tažením po ploše *layoutu*. Další úpravy obsahu, mazání a kopírování je dostupné z kontextové nabídky objektu *LayoutText*, která je vyvolána kliknutím pravého tlačítka na daném objektu.

Všechny popisky *LayoutText* v *layoutu* jsou udržovány, podobně jako *LayoutItem*, v seznamu *MlstTextsList*. Ten je při každé změně podoby *layoutu* v metodě *MainCanvasRefresh* stejně, jako je to u *LayoutItem*.

Výběr objektů v *layoutu*

Jak bylo uvedeno výše, v event handlerech *MainCanvasMouseMove*, *MainCanvasMouseLeftButtonDown* a *MainCanvasMouseLeftButtonUp* je implementována mimo jiné i logika výběru prvků na *layoutu*. Oblast výběru je zde znázorněna obdélníkem (*System.Windows.Shapes.Rectangle*), který je dle potřeby přidáván nebo odebírá z prvku *MainCanvas*.



Obrázek 27: Ukázka výběru prvků v *layoutu*

Při vytváření výběrového obdélníku zde bylo problémem identifikace prvku pod kurzorem v momentě stisknutí levého tlačítka myši (počátek výběru). Požadovanou funkcí totiž bylo:

- Při kliku do volného prostoru mezi prvky se zruší aktuální výběr a zahájí se výběr pomocí výběrového obdélníku.
- V případě, že bylo levé tlačítko stisknuto na již vybraném objektu, musí dojít k posunu celého výběru o daný vektor.
- V případě stisknutí tlačítka na nevybraném prvku je zahájen posun pouze prvku pod kurzorem.

Tento problém řeší třída *System.Windows.Media.VisualTreeHelper*, která dokáže pracovat se stromem vizuálních prvků WPF (*System.Windows.Media.VisualTree*), pomocí metody *HitTest* určovat tzv. *Visual* (tedy jakýkoliv viditelný grafický prvek), který je v hierarchii *System.Windows.Media.VisualTree* jako poslední nacházející se pod daným bodem (v našem případě kurzorem), ale také určovat potomky či rodiče ve *System.Windows.Media.VisualTree*.

Na základě metody *VisualTree.HitTest* se tedy určí, zda jde o *LayoutItem* (*LayoutText*) nebo o prázdný prostor. Pokud se jedná o *LayoutItem* (*LayoutText*), rozhodne se další logikou o příslušnosti do předchozího výběru a provedou se zmíněné operace.

```
1  bool hit = false;
2  bool hit2 = false;
3  try
4  {
5      FrameworkElement o =
6          (FrameworkElement) VisualTreeHelper.HitTest(MainCanvas, _startDrag).VisualHit;
7      FrameworkElement parent = (FrameworkElement) ((FrameworkElement) o.Parent).Parent;
8      if (MdevDragDevice.Contains(parent) || MdevDragTexts.Contains(parent)) hit = true;
9
10     if (
11         ((FrameworkElement) o.Parent).Parent.GetType() == typeof (LayoutItem) ||
12         ((FrameworkElement) o.Parent).Parent.GetType() == typeof (LayoutTexts)
13     )
14     { hit2 = true;}
15 }
16 catch
17 {
18 }
```

Obrázek 28: Ukázka kódu z event handleru *MainCanvasMouseLeftButtonDown*

Obrázek 28 obsahuje ukázkou zdrojového kódu z event handleru *MainCanvasMouseLeftButtonDown*, který rozhoduje o identifikaci prvku. Na základě proměnných *hit* a *hit2* se určují nastalé situace. Pokud jsou obě proměnné *false*, tak se jedná o prázdné místo. Pokud proměnná *hit* nabude hodnoty *false* a *hit2 true* jedná se o neoznačený objekt.

Prvky *layoutu* jsou poté vybírány dle příslušnosti pozice x a y do dané oblasti výběrového obdélníku. Pro realizaci výběru můžeme použít opět výhod komponenty LINQ (*LINQ to Object*). Obrázek 29 obsahuje konkrétní realizaci výběru:

```

1 MdevDragDevice = MlstDeviceList.Where(it => it.OnlineX > x &&
2 it.OnlineY > y &&
3 it.OnlineX < x + _selectRectangle.ActualWidth &&
4 it.OnlineY < y + _selectRectangle.ActualHeight).ToList();

```

Obrázek 29: Realizace výběru objektů v *layoutu*

Výběrový obdélník je *_selectRectangle*, *it* je *LayoutItem* v seznamu prvků na *layoutu* a x a y jsou počáteční souřadnice výběrového obdélníku (stejná metoda je použita u objektů typu *LayoutText*). Pokud se prvek v oblasti nachází, je zařazen do výběrového seznamu *MdevDragDevice*, se kterým je možné pracovat samostatně. Uvědomme si, že do tohoto seznamu jsou předávány pouze reference na tyto vybrané prvky, nejedená se tedy o zvýšení paměťové náročnosti.

5.2.1.3 Implementace tabulkového pohledu na layout

Z analýzy vyplývá, že aplikace musí nabízet také tabulkové rozhraní, kde by bylo možné přehledně editovat, vyhledávat a odebírat stroje z *layoutu*. V tomto případě je tento tabulkový náhled implementován prvkem *System.Windows.Controls.DataGrid*, konkrétně v XAML dokumentu je pojmenován *RowView*.

WVKCTRID	Tvar	Pozice X	Pozice Y
WVKCTRID 1	Machine 4	30	170
WVKCTRID 1	Machine 4	140	170
WVKCTRID 1	Machine 4	30	260
WVKCTRID 1	Machine 4	140	260
WVKCTRID 2	Machine 4	384	171
	Machine 4	491	171
	Machine 4	390	266
	Machine 4	491	266
	Machine 3	740	170
	Machine 3	740	170
	Machine 3	800	270
	Machine 3	800	170
	Machine 3	930	270
	Machine 3	930	170
	Machine 3	990	270
	Machine 3	990	170
	Machine 3	1150	270
	Machine 3	1150	170
	Machine 3	1210	270
	Machine 3	1210	170

Obrázek 30: Náhled tabulkového rozhraní

Tento prvek je ve výchozím stavu skrytý a samotné přepnutí do tabulkového rozhraní je realizováno jeho zviditelněním. Tím dojde k překrytí prvku pro grafickou úpravu *layoutu*. Zároveň se zviditelněním *RowView* se v nástrojové liště zobrazí prvek pro vyhledávání v tabulkovém rozhraní (Obrázek 30 - prvky označené červeným obdélníkem). Ten umožňuje filtrovat zobrazované řádky podle *WRKCTRID*.

Pro provázání tabulky se seznamem prvků v *layoutu* (*MlstDeviceList*) je využito obousměrného databindingu. Což znamená, že pokud dojde ke změně hodnoty v tabulce nebo v položce seznamu *MlstDeviceList*, změna se projeví okamžitě na obou místech.

Buňky jsou nabídnovány na atributy třídy *LayoutItem*. Odlišná je buňka ve sloupci *Shape* (Tvar), ta je nadefinována jako *ComboBox*, aby bylo možné vybírat pouze z dostupných *shape*. *ComboBox* je ve výchozím stavu nastaven na jméno aktuálního tvaru. Samozřejmě po výběru jiného tvaru je objekt *LayoutItem* aktualizován.

Pro filtrování a vyhledávání v tabulce je využito rozhraní *ICollectionView*, které umožňuje definovat pravidla pro filtrování, řazení a další operace pro zobrazení dat v prvcích WPF pracujících se seznamy. Změna filtru je aktivována při každé změně obsahu (event *TextChanged*) vyhledávacího pole. To zvyšuje míru interaktivity s uživatelem a urychluje vyhledávání. Při spouštění vyhledávání bylo nutné zajistit, aby buňky nebyly v editačním módu. Proto je zde použita funkce *CommitEdit()*, která by měla zajistit potvrzení probíhajících změn a vystoupení z editačního módu buňky. Použití filtru je velice jednoduché (Obrázek 31):

```
1 ICollectionView view = CollectionViewSource.GetDefaultView(MlstDeviceList);
2 view.Filter = delegate(object item)
3     {
4         bool match = ((LayoutItem) item).WrkctrId.Contains(SearchTextBox.Text);
5         return match;
6     };
```

Obrázek 31: Použití filtru u *DataGrid*

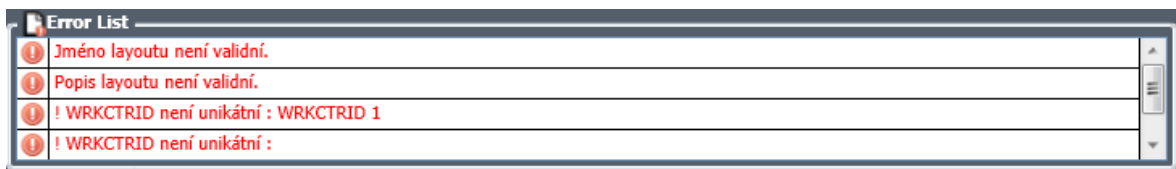
Filtr je nadefinován anonymní metodou, která určuje, zda atribut *WRKCTRID* obsahuje hledaný výraz.

Zajímavou funkcí (zejména pro uživatele) je propojení výběru prvků v tabulce s výběrem prvků v grafické editaci. Prakticky tato funkce vypadá tak, že pokud uživatel vybere prvky v tabulkovém rozhraní a přepne se do grafického rozhraní, jsou tyto prvky označeny a je možné s nimi dále pracovat, jakoby selekce proběhla v grafickém rozhraní.

Tabulkové rozhraní nezahrnuje objekty typu *LayoutText* (tedy popisku v *layoutu*), které jsou v *layoutu* minoritní, a tedy není potřebná jejich hromadná a pohodlnější editace.

5.2.1.4 Zobrazování chyb a jejich zjišťování

Chyby v *layoutu* jsou zobrazovány v *DataGridErrorListBox*, jsou zde jak chyby atributů *layoutu*, tak chyby způsobené duplicitou atributu *WRKCTRID* u různých *LayoutItem*. U každé chyby je možné dvojklikem zobrazit dialog pro úpravu chybného atributu, nebo v případě duplicity *WRKCTRID* označit *LayoutItem* s touto chybou. K funkcionalitě mne inspirovala mnohá vývojová IDE, kde se po dvojkliku zobrazí řádek s chybou, popřípadě otevře příslušný dialog.



Obrázek 32: Detail prvku pro zobrazení chyb

Chyby jsou uchovávány v seznamu *_mlstError*. Tento seznam je použit jako zdroj dat právě pro výše zmiňovaný *ErrorListBox*. Pro zobrazení chyby je v *ErrorListBoxu* použito vlastní šablony pro dosažení požadovaného vzhledu (Obrázek 33) (tady vidíme, že technologie WPF umožňuje opravdu jednoduché vytvoření požadovaného vzhledu grafických prvků).

```

1 <DataGrid.Columns>
2   <DataGridTemplateColumn>
3     <DataGridTemplateColumn.CellTemplate>
4       <DataTemplate>
5         <Image Source="Images/exclamation.png"/>
6       </DataTemplate>
7     </DataGridTemplateColumn.CellTemplate>
8   </DataGridTemplateColumn>
9   <DataGridTextColumn Width="*" Binding="{Binding Path=Error}" Foreground="Red" />
10 </DataGrid.Columns>

```

Obrázek 33: Definice podoby řádku *ErrorListBox* v XAML

Kontrolu unikátnosti *WRKCTRID* si popíšeme v následujícím odstavci. Pro kontrolu validnosti *layoutu* je napsána funkce přímo v třídě *LayoutClass*, která kontroluje obsah jednotlivých atributů a bude zmíněna v sekci pro popis datových tříd.

Na objekty popisků *layoutu LayoutText* se nevztahují žádná omezení, a proto u nich není žádným způsobem prováděna kontrola.

Kontrola unikátnosti *WRKCTRID* v *layoutu*

Zajímavým problémem se ukázala kontrola unikátnosti atributu *WRKCTRID* u *LayoutItem* v *layoutu*. Pokud by v *layoutu* došlo k duplicitnímu výskytu hodnoty atributu, duplicitnost musí být indikována v seznamu chyb. V tomto případě jsem se rozhodl využít komponenty *LINQ to Object* a použít ke kontrole unikátnosti jednoduchý algoritmus. Nejdříve proběhne selekce jednotlivých *WRKCTRID* bez duplicit (Obrázek 34).

```
1 List<string> uniqNames = (from item in MlstDeviceList
2   select item.WrkctrId).Distinct().ToList();
```

Obrázek 34: Selekcce jmen při kontrole unikátnosti *WRKCTRID*

Následně se pomocí funkce *DoValidate* ověří počet výskytů *LayoutItem* s daným *WRKCTRID*. Pokud je takovýchto víc, jejich výběr je uložen do struktury *ErrorItem* (testování na unikátnost v následujícím zdrojovém kódu). Uložení výběru je zde z důvodu pozdějšího zobrazení chyb (Obrázek 35). Pokud by se totiž ukládalo jen *WRKCTRID* pro identifikaci duplicit, muselo by při další interakci s grafickou podobou chyby docházet k opětovnému vyhledávání objektů v seznamu *MlstDeviceItem*. V tomto případě to ale není třeba, jelikož výběr prvků je již uložen v struktuře pro uchování chyby.

```
1 private ErrorItem DoValidate(string name){
2   List<LayoutItem> errorItems = (from item in MlstDeviceList
3     where item.WrkctrId == name
4     select item).ToList();
5
6   if (errorItems.Count > 1){
7     return (new ErrorItem("!" + Texts.Warning_WRKCTRID + name,
8       errorItems.ToArray()));
9   }
10
11   return null;
12 }
```

Obrázek 35: Implementace vyhledávání duplicit *WRKCTRID*

5.2.1.5 Implementace Undo/Redo

Jak již bylo uvedeno v návrhu aplikace v sekci (3.3), pro realizaci funkcionality jsou dva limitované zásobníky. Jelikož žádná třída ze jmenného prostoru *System.Collections* nenabízí kolekci, kde by bylo možné napevno nastavit počet položek tak, aby po přidání položky při maximálním počtu položek v kolekci byla nejstarší položka zahozena, bylo

nutné implementovat vlastní strukturu *LimitedStack*, která bude popsána v sekci (5.4.4). V programové realizaci jsou zásobníky v podobě proměnných *_UndoBuffer* a *_RedoBuffer*. Zásobníky jsou limitovány na deset stavů, což je dle mého soudu rozumný počet. Ukládaný stav je hluboká kopie objektu *Layout*, která reprezentuje aktuální podobu *layoutu*, jak jsem uvedl v sekci (5.2.1.2).

Jádrem funkcionality je funkce *UpdateBuffer* (potažmo funkce *UpdateUndoBuffer*), která zajišťuje vytvoření hluboké kopie a uložení kopie na příslušný zásobník. Tato funkce je spouštěna při každé aktivitě měnící stav *layoutu* (změna atributů, přidání *LayoutItem* (*LayoutText*) apod.).

Pokud uživatel použije funkce *Undo* (*Redo*), je z příslušného event handleru volána funkce *UndoRedoAction*, ta zajišťuje vytažení předcházejícího stavu z příslušného zásobníku a nastavení aktuálního stavu a zároveň uložení stávajícího stavu na druhý zásobník. Při použití funkce *Undo* a následném pokračování v úpravách *layoutu* je obsah zásobníku *_RedoBuffer* vymazán, jelikož jeho obsah je z logického hlediska při další úpravě neplatný.

5.2.1.6 Implementace Snap to grid a funkce ORTHO

Funkcionalita „*Snap to grid*“ je z velké části implementována v event handleru *MainCanvasMouseMove*. Zapnutí této funkce je indikováno příznakem *_snapGridEnabled*. Při pohybu myši po ploše *MainCanvas* je tento příznak testován a přemísťování objektů se děje pouze v „diskrétních“ krocích (v tomto případě pouze po 10 pixelech). Pro orientaci uživatele je při tomto aktivním příznaku v metodě *MainCanvasRefresh* vykreslována mřížka v podobě prvku *SnapGrid* odvozeného od třídy *Canvas*.

Funkce *ORTHO*, tedy pohyb pouze ve vodorovném nebo svislém směru je realizována podobným způsobem. Při pohybu myši je testován stav klávesy „Shift“. Pokud je tato klávesa stisknutá, pohyb myši se omezí pouze na pohyb po jedné ze souřadnic. Rozhodnutí o tom, po které souřadnici se má objekt pohybovat, je realizováno porovnáním velikosti vektorových složek. Pohyb se pak uskutečňuje po souřadnici, u níž byla vektorová složka větší.

5.2.1.7 Implementace otevírání/ukládání do souboru

Uložení a načtení souboru je implementováno v metodách *Serialize* a *Deserialize*. Obě tyto funkce využívají pro vytvoření nebo načtení XML struktury *layoutu* komponentu *LINQ to*

XML, která velice usnadňuje práci s XML dokumentem. Pro uložení do souboru byla vytvořena následující struktura (Obrázek 36):

```

1  <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2  <SaveContent>
3    <Devices>
4      <LayoutItem Id="0" LayoutId="0"
5        LCD_Scale="0" Monitor_Scale="50"
6        Online_X="261" Online_Y="161"
7        ShapeId="3" WRKCTRID=""
8      />
9    </Devices>
10   <Layout Id="0" InsUser="User"
11     InsDate="0001-01-01T00:00:00"
12     Name="" OnlineHeight="0"
13     OnlineWidth="0" Description=""
14     Layout=""
15     ValidFrom="2012-03-28T00:00:00+02:00"
16     ValidTo="2013-03-28T00:00:00+01:00" IsWebLayout="false"
17   />
18 </SaveContent>

```

Obrázek 36: XML struktura pro uložení *layoutu* do souboru

Z této struktury je vidět, že kořenový element *SaveContent* obsahuje dva hlavní elementy *Devices* a *Layout*. Element *Devices* obsahuje uložený seznam *LayoutItem*, tedy XML kopii seznamu *MlstDeviceList*, a element *Texts* obsahuje seznam objektů *LayoutText*, zatímco element *Layout* obsahuje XML kopii ukládaného *layoutu*. V programu je pak možné načíst pomocí konstrukce LINQ kompletně celý seznam strojů a data o *layoutu*.

5.2.1.8 Implementace otevírání/ukládání *layoutu* do databáze

Prohlížení obsahu databáze je realizováno pomocí dialogového okna *LayoutBrowserDialog*, který je popsán v sekci (5.2.2). V tomto dialogu je v public property k dispozici vybrané *id*, které se stává parametrem metody *SqlDeserialize*.

Pro samotné ukládání/načítání *layoutu* z databáze jsou implementovány funkce *SqlSerialize* a *SqlDeserialize*. O většinu práce se zde stará opět komponenta *LINQ to SQL*.

Důležitým faktem je, že funkce *SqlSerialize* před uložením *layoutu* do databáze kontroluje, zda v databázi již neexistuje *layout* se stejným *id*, jako je u ukládaného *layoutu*. Pokud v databázi takový *layout* existuje, dojde pouze k aktualizaci stávajícího *layoutu*, aby bylo zachováno původní *id* *layoutu*.

5.2.1.9 Implementace funkcionality *AutoSave*

Funkcionalita *AutoSave*, tedy automatická záloha aktuálně rozpracovaného *layoutu*.

Layout je ukládán do dočasných souborů, jejichž cesty jsou získány při inicializaci hlavního okna (tedy vlastně i aplikace). Tento seznam je uložen ve statické třídě *AutoSave* (5.4.3). Automatické ukládání v intervalech 15 minut je realizováno časovačem, který spouští ukládání v event handleru *TimerTick*. Tento event handler zařazuje do hlavní smyčky aplikace požadavek na spuštění funkce *Autoserialize*, ve které je implementováno samotné získání cesty k dočasnému souboru a uložení.

Pro tuto funkcionalitu byla zvolena nízká priorita zpracování, a tedy je požadováno spuštění funkce *Autoserialize* v době, kdy je aplikace ve stavu *idle* (tedy na pozadí aplikace).

5.2.1.10 Klávesové zkratky

Klávesové zkratky napříč celým hlavním oknem jsou realizovány prostřednictvím tzv. *Commands* (příkazy), což jsou v podstatě události, ale jejich použití je mnohem širší. V tomto případě je využito příkazů právě pro jejich univerzálnost. Umožňují totiž spuštění totožné logiky příkazu několika cestami. To znamená, že například akce *Copy*, může být spouštěna jak z kontextového menu *LayoutItem*, tak pomocí klávesové zkratky a také z hlavního menu, to vše pouhým provázáním daného prvku s příslušným příkazem.

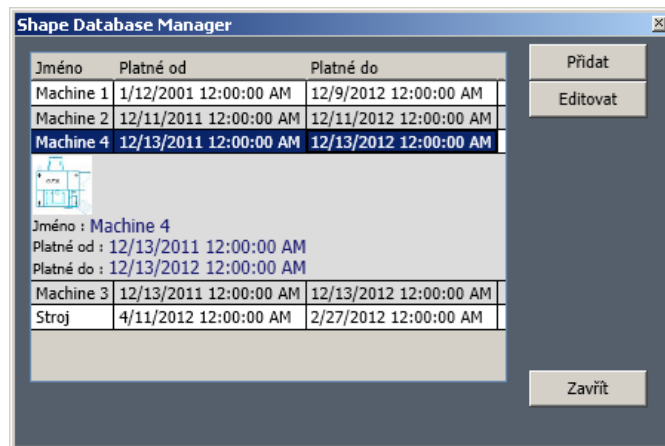
Klávesové zkratky jako takové jsou v XAML souboru definovány třídou *System.Windows.Input.KeyBinding*, která svazuje právě klávesovou zkratku s příslušným příkazem. Příkladem může být provázání zkratek u grafického prvku *MainScrollViewer* (Obrázek 37):

```
1 <ScrollViewer.InputBindings>
2   <KeyBinding Command="Copy" Key="C" Modifiers="Ctrl"/>
3   <KeyBinding Command="Paste" Key="V" Modifiers="Ctrl"/>
4   <KeyBinding Command="Cut" Key="X" Modifiers="Ctrl"/>
5   <KeyBinding Command="Delete" Key="Delete"/>
6 </ScrollViewer.InputBindings>
7 <ScrollViewer.CommandBindings>
8   <CommandBinding Command="Copy" Executed="MainCanvasCopyExecuted"/>
9   <CommandBinding Command="Delete" Executed="MainCanvasDeleteExecuted"/>
10  <CommandBinding Command="Paste" Executed="MainCanvasContextPasteClick"/>
11  <CommandBinding Command="Cut" Executed="MainCanvasContextCutClick"/>
12 </ScrollViewer.CommandBindings>
```

Obrázek 37: Definice *Commands* u *ScrollViewer*

5.2.2 LayoutBrowserDialog a ShapeBrowserDialog

Tato třída implementuje dialog pro procházení databáze *layoutů* (*shape*) (dále jen záznamy). Jeho hlavní součástí je *DataGrid*, jehož *ItemSource* je při inicializaci nabídnováno na záznamy v databázi.

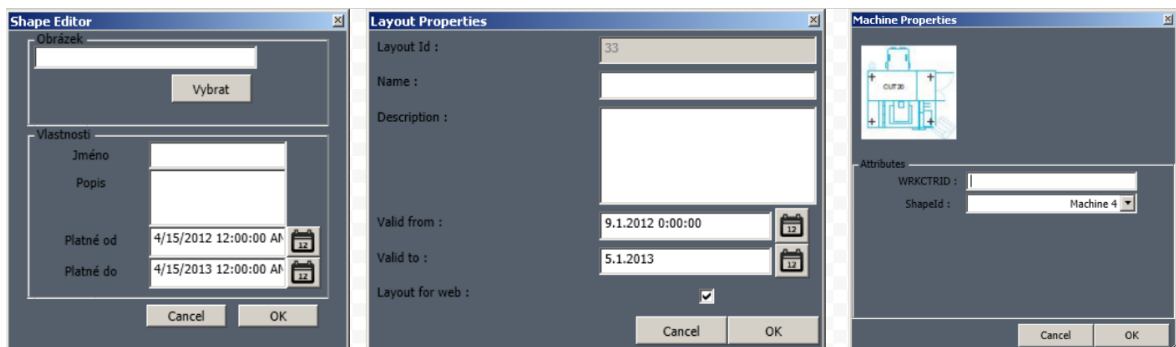


Obrázek 38: Dialog pro prohlížení databáze

Významným prvkem je detailní pohled na vybraný řádek. Tato funkcionality je součástí implicitních funkcionalit třídy *DataGrid*, nicméně musí být explicitně nadefinována v souboru XAML, který je součástí definice dialogu. Náhled je definován v atributu *RowDetailsTemplate* třídy *DataGrid*.

5.2.3 LayoutMgr, ShapeMgr, TextMgr a LayoutItemMgr

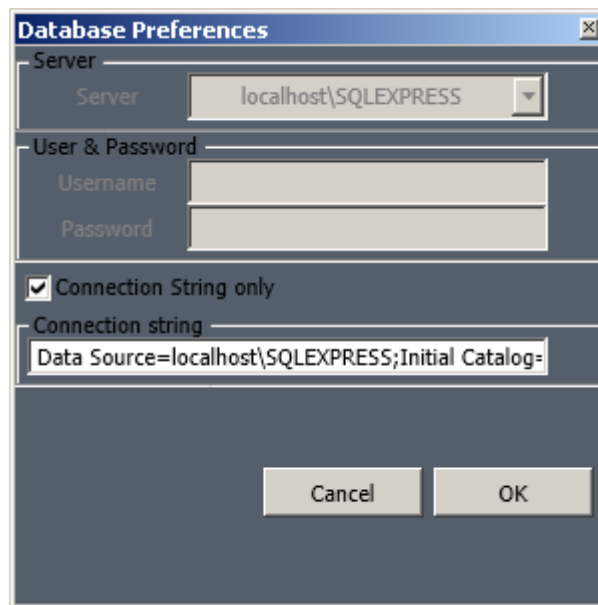
Všechny tyto tři třídy jsou si velice podobné. Atributy jednotlivých instancí jsou nabídnovány na textboxy. V konstruktoru dialogu je vždy předána reference na upravovaný objekt. Po potvrzení dialogu jsou příslušné atributy změněny.



Obrázek 39: Ukázky dialogů *LayoutMgr*, *ShapeMgr* a *LayoutItemMgr*

5.2.4 PreferencesDialog

Třída *PreferencesDialog* implementuje dialog pro nastavení připojení k databázi. Je zde možnost nastavení IP adresy nebo *hostname* serveru a přihlašovacích údajů. Pro rychlejší nastavení je zde také možnost zadat tzv. *Connection String*. Po zavření dialogu je aktualizován seznam *shape* v *MainListBoxu*.



Obrázek 40: *PreferencesDialog*- dialog pro nastavení připojení k databázi

5.3 Třídy DATA & LINQ

Všechny tři třídy mají namapovány atributy na příslušné tabulky v databázi, aby bylo zajištěno obousměrné propojení atributů se záznamy v databázi a zároveň aby bylo umožněno obousměrné bindování objektů mají všechny tři třídy implementováno rozhraní *INotifyPropertyChanged*. To zajišťuje, že u všech objektů svázaných (nabindovaných) s příslušným atributem dojde při jeho změně k aktualizaci hodnoty tohoto atributu.

5.3.1 LayoutClass

Ve spojení s řešením funkcionality *Undo/Redo* bylo nutné vytvořit funkci pro vytvoření hluboké kopie objektu. Jako obecně nejlepší řešení se k vytvoření hluboké kopie používá serializace a následná deserializace objektu. Toto implementuje funkce *DeepCopy*, jejíž

návratová hodnota je právě nový objekt typu *LayoutClass* s totožnými atributy jako má instance, která metodu zavolala.

Další důležitou metodou v této třídě je metoda *Validate* realizující procedurální integritní omezení. Aby byl *layout* validní pro uložení do databáze, musí obsahovat jméno, jeho datum musí dodržovat časovou posloupnost a musí obsahovat pozadí (plán haly).

```
1 public void Validate() {
2     this.Errors.Clear();
3     bool name = (_name.Trim().Length > 0);
4     bool description=false;
5     if(!string.IsNullOrEmpty(_description))description=true;
6     bool date = (_dateFrom < _dateTo);
7     bool background = (Layout.Length > 0);
8     _valid = name & date & background;
9
10    if (!_valid | !description) {
11        if (!name)
12            Errors.Add(new ErrorItem(Texts.Warning_LayoutName));
13        if (!description)
14            Errors.Add(new ErrorItem(Texts.Warning_LayoutDescription));
15        if (!date)
16            Errors.Add(new ErrorItem(Texts.Warning_LayoutDate));
17        if (!background)
18            Errors.Add(new ErrorItem(Texts.Warning_LayoutBackground));
19    }
20 }
```

Obrázek 41: Implementace metody *Validate* třídy *LayoutClass*

Z kódu (Obrázek 41) je patrné, že pokud nejsou dodržena integritní omezení, jsou v objektu ukládány chyby, které jsou nadále zobrazovány v hlavním okně.

5.3.2 ShapeClass

Podobně jako *LayoutClass* obsahuje třída *ShapeClass* prostředek implementující kontrolu integritních omezení. V tomto případě jsem se rozhodl realizovat kontrolu pomocí property *Valid*, které je jen pro čtení a jeho accesor *get* obsahuje proceduru kontrolující obsah pojmenování *shape* a obsah atributu *ShapeImage* (grafické reprezentace přiřazené k tvaru). Návratovou hodnotou je typ *boolean* indikující validnost *shape*.

5.3.3 LayoutItem

Třída *LayoutItem* je odvozena od třídy *UserControl* pro jednoduché zacházení s touto třídou při implementaci grafického rozhraní. Nicméně ve spojitosti s použitím komponenty LINQ pro namapování třídy na tabulku v databázi toto přináší i svoje úskalí, zejména nutnost implementace rozhraní *ISerializable* pro definici vlastní serializace (jako u

LayoutClass součástí metody pro vytvoření hluboké kopie), která je pro třídy odvozené od *UserControl* bez vlastní implementace nepoužitelná.



Obrázek 42: Grafická podoba *LayoutItem* v *layoutu*

5.3.4 LayoutText

Třída *LayoutText* je stejně jako třída *LayoutItem* odvozena od základní třídy *UserControl*. Třída je namapována na tabulku databáze s názvem *LayoutTexts*. Pro účely funkcionality *Undo/Redo* je implementováno rozhraní *ISerializable*. Specialitou třídy je nabindování grafických prvků na property *TextCz*, *TextEn* a *Number*, které reprezentují jednotlivé části popisky v *layoutu*.

123/ABC(1234ABCD)

Obrázek 43: Grafická podoba *LayoutText* v *layoutu*

5.4 Třídy HELPER

5.4.1 DataSourceClass

Statická třída *DataSourceClass* obsahuje pomocné statické proměnné pro uchování hodnot nastavení, načítání a ukládání konfiguračního souboru, ukládání a načítání offline i online knihovny dat. Dále jsou zde metody pro logování chyb programu a pomocné metody pro konverzi tříd a souborů do požadovaných datových typů.

Metody pro logování chyb jsou důležitou součástí při testování aplikace. Při selhání aplikace se ve speciálním systémovém adresáři *AppData* (*Data aplikací*) a podadresáři *Layout Designer* vytvoří logovací soubor *errorlog.txt* a zapíše se do něj záznam chyby ve specifickém formátu (Obrázek 44).

```
1 << Datum a čas >>
2 << Stack Trace >>
3 << Message >>
```

Obrázek 44: Formát záznamu v logovacím souboru

Pouhým nahlédnutím do tohoto logovacího souboru pak je velice snadné odhalit původ chyby nebo dokonce pádu aplikace.

5.4.2 ErrorItem

Třída *ErrorItem* je objektem zapouzdřujícím obsah chyby, popřípadě výběr *LayoutItem* z *layoutu*, které tyto chyby obsahují. Skládá se z property *Error* popisující chybu slovně a dále tedy ze seznamu *ErrorItems*, který je volitelně naplněn objekty obsahujícími chybu.

5.4.3 AutoSave

Statická třída *AutoSave* navenek reprezentuje statický kruhový seznam dočasných souborů, do kterých je průběžně ukládána záloha právě rozpracovaného *layoutu*. Jádrem třídy je funkce *GetFile*, která vrací aktuální jméno souboru, do kterého má být automatická záloha zapsána, zároveň ověřuje a zajišťuje funkčnost kruhového seznamu.

5.4.4 LimitedStack

Třída *LimitedStack* implementuje velikostně limitovaný zásobník. Jak jsem zmínil v sekci (5.2.1.5), .NET Framework nenabízí kolekci, která by striktně omezovala počet položek v kolekci a při překročení limitu nejdříve vloženou položku odstranila (tedy LIFO). Byl jsem nucen implementovat si kolekci vlastní.

Kolekce je odvozena od kolekce *LinkedList*, navíc obsahuje proměnnou udávající limit počtu objektů v kolekci. Pro funkci zásobníku jsou implementovány metody *Pop*, *Push*, *Peek*. V metodě *Push* je pak kontrolováno dosažení limitu (Obrázek 45).

```
1 public void Push(T value)
2     {
3         LinkedListNode<T> node = new LinkedListNode<T>(value);
4         this.AddLast(node);
5
6         if (this.Count > _maxItems)
7             {
8                 this.RemoveFirst();
9             }
10    }
```

Obrázek 45: Implementace metody *Push* v třídě *LimitedStack*

5.4.5 Clipboard

Statická třída *Clipboard* implementuje vlastní funkčnost schránky. Bylo by sice možné, využít systémové schránky, ale museli bychom vytvořit strukturu pro uložení potřebných

informací, jelikož do schránky chceme uložit jak seznam jednotlivých *LayoutItem*, tak případně souřadnice na kterých bylo vložení do schránky zavoláno.

Stěžejními metodami jsou metody *InsertItems* a metoda *GetItemsData*, ty zajišťují vkládání a vybírání dat ze schránky. Seznam aktuálních *LayoutItem* ve schránce je umístěn ve statické proměnné *_mlst_copyItems*, ta obsahuje hlubokou a tedy neměnnou kopii vložených dat. Další důležitou proměnnou je zde proměnná, potažmo property *startPoint*, které obsahuje místo vzniku volání příkazu *Copy*. V případech spuštění přes kontextové menu je poté při vkládání od tohoto bodu počítán vektor posunutí, tedy místo umístění kopie. V případě využití klávesových zkratk je vektor posunutí pevně daný.

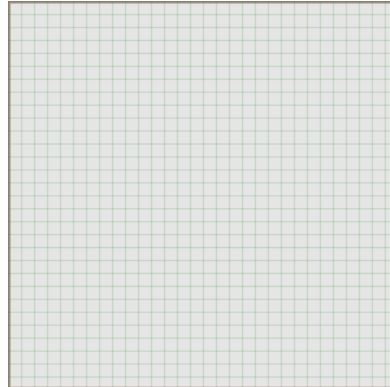
5.4.6 SnapGrid

Třída *SnapGrid* je odvozena od základní třídy *Canvas* a slouží pouze jako vizuální pomůcka uživatele, při zapnutí funkcionality „*Snap to Grid*“. Jako zajímavé řešení pro zobrazení mřížky je definice výplně prvku *SnapGrid* (Obrázek 46).

```
1 <Canvas.Background>
2   <DrawingBrush Viewport="0,0,10,10"
3     ViewportUnits="Absolute"
4     TileMode="Tile"
5     Opacity="0.6">
6     <DrawingBrush.Drawing>
7       <DrawingGroup>
8         <GeometryDrawing
9           Geometry="M0,0 L1,0 1,0.02, 0,0.02Z" Brush="Green" />
10        <GeometryDrawing
11          Geometry="M0,0 L0,1 0.02,1, 0.02,0Z" Brush="Green" />
12        </DrawingGroup>
13      </DrawingBrush.Drawing>
14    </DrawingBrush>
15  </Canvas.Background>
```

Obrázek 46: Definice výplně pozadí *UserControl* prvku *SnapGrid*

Na řádku devět a jedenáct je vidět ukázka tzv. „mini-language“, který nám dovoluje popsat vykreslovanou geometrii v XAML velice kompaktně a vyhnout se tak zbytečnému nabývání objemu kódu. Zde je geometrie pozadí dlaždice (*Tile*) složena ze dvou lomených čar tvořících čtverec. Výsledkem je pozadí imitující milimetrový papír (Obrázek 47).



Obrázek 47: Výsledné pozadí vytvořené pomocí mini-language

5.5 Lokalizace aplikace

Lokalizace aplikace je v tomto případě realizována pomocí zdrojů ve formátu *resx*. V konstruktoru hlavního okna aplikace je určeno jazykové prostředí (pomocí třídy *CultureInfo.CurrentCulture*) ve kterém se aplikace nachází. V tomto případě se rozlišuje české a cizojazyčné prostředí (v němž je použita anglická verze lokalizačního souboru). Nastavením verze zdrojového souboru (nastavení property *Culture*) jsou poté v aplikaci volány řetězce ze souboru *Texts.resx* v patřičné jazykové verzi.

Výhodou tohoto řešení je velmi jednoduchá lokalizace do dalších jazyků. Ta by v případě potřeby zahrnovala pouze přeložení lokalizačního souboru *Texts.resx* do požadovaného jazyka.

6 DEPLOYMENT APLIKACE

Tato část práce popisuje všechny úkony, které bylo nutné provést pro finální deployment aplikace, který obnáší jednak vytvoření instalační sady přes tvorbu uživatelského manuálu a finální úpravy aplikace pro umožnění běžné instalace a používání.

6.1 Tvorba uživatelského manuálu

Jednou ze součástí nasazení aplikace byla tvorba uživatelského manuálu, který by měl popisovat všechny funkcionality aplikace a provést uživatele postupně tvorbou *layoutu*.

6.1.1 Zaměření uživatelského manuálu

Při tvorbě manuálu bylo nutné zvážit cílovou skupinu, na kterou je manuál zaměřen, a od tohoto určení odvodit jazykové a výrazové prostředky a odbornost popisu aplikace.

V tomto případě byl cílovou skupinou běžný uživatel PC ovládající kancelářské aplikace a se základními až pokročilými znalostmi užívání počítače. V souvislosti s tímto zaměřením byla zvolena cesta grafického průvodce s velkým zastoupením názorných snímků aplikace a případným komentářem o prováděných akcích nebo alternativním postupu prováděné akce.

6.1.2 Koncepce manuálu

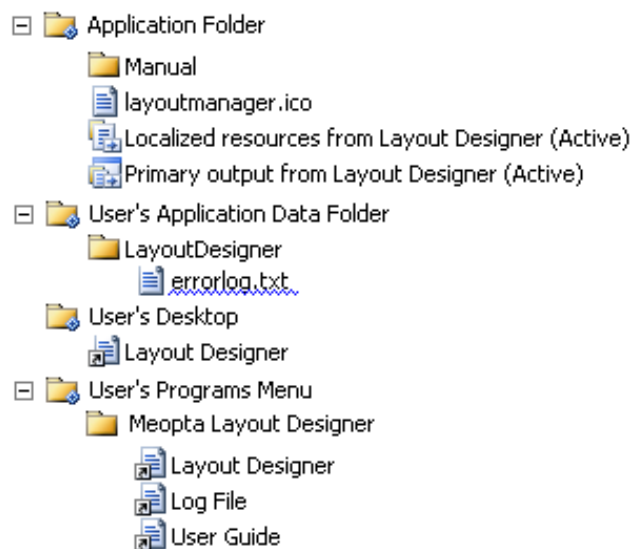
Samotný obsah manuálu je koncipován tak, aby uživatele postupně prováděl tvorbou *layoutu* s následnou správou *layoutů* a položek v databázi. Popisuje založení nového *layoutu*, přidání jednotlivých *LayoutItem* a *LayoutText* do *layoutu*, definici jejich atributů, přemísťování objektů a možnosti editace. Dále jsou v *layoutu* uvedeny všechny klávesové zkratky, základní popis prvků hlavního okna aplikace a jejich chování. V neposlední řadě je zde popisována oprava chyb v *layoutu*, možnosti identifikace chyb a možná chybová hlášení.

6.2 Instalační balíček

Pro snadnou instalaci aplikace na cílový stroj bylo vhodné vytvořit instalační balíček, který provede standardní instalaci v rámci operačního systému tak, aby i odinstalace, popřípadě reinstalace software proběhla bez potíží a zbytečného dohledávání jeho součástí.

6.2.1 Vytvoření instalačního balíčku MSI

K vytvoření instalačního balíčku bylo použito editoru ORCA pro balíčky MSI, který je součástí vývojového prostředí Microsoft Visual Studio. Ten poskytuje poměrně dobrý nástroj k přípravě instalačního balíčku včetně definice vytvoření podsložek, jejichž umístění je specifické v prostředí operačního systému konkrétně složky pro data aplikací zmiňované v kapitole (5.2.1.1) a složky pro dočasné soubory aplikací popisované v kapitole (5.2.1.9) .



Obrázek 48: Struktura MSI package pro deployment aplikace

Ze struktury instalačního balíčku (Obrázek 48) je patrné, že balíček již zahrnuje soubor uživatelského manuálu. Během instalace je také vytvořen adresář v nabídce Start, v tomto adresáři jsou dostupné odkazy na všechny potřebné soubory. Zároveň je vytvořena ikona na pracovní ploše, odkud je možné program spouštět.

6.2.2 Otestování instalačního balíčku

Po vytvoření balíčku bylo vhodné balíček otestovat. Jako vhodný prostředek se jeví nejprve instalaci otestovat na virtuálním stroji, aby případně nedošlo k jakémukoliv poškození živého systému.

ZÁVĚR

V rámci této práce byla realizována desktopová aplikace od analýzy požadavků a jejího návrhu, až po kompletní implementaci, otestování a deployment aplikace se všemi náležitostmi, ke kterým patří uživatelský manuál a instalační balíček aplikace.

Při analýze požadavků byl nejdříve proveden sběr požadavků na základě oboustranného dialogu řešitele se zadavatelem a následně vytvořen dokument, který tyto požadavky shrnoval. Dle tohoto dokumentu byly analyzovány jednotlivé funkční a nefunkční požadavky a v případě nejasností byly doplněny upřesňující informace. Požadavky byly jasně identifikovány v balíčkovém diagramu požadavků a na jejich základě byl vypracován use-case model, který měl zachycovat spolupráci systému s uživatelem. V rámci tohoto modelu byly rozpracovány scénáře případu užití pro jasnější představu interakce uživatele se systémem.

Po těchto krocích byl proveden návrh aplikace po stránce grafické a logické. Tento krok zahrnoval návrh podoby jednotlivých dialogů, ovládacích prvků a funkcí týkajících se ovládání aplikace.

Samotná implementace se odehrávala ve čtyřech hlavních fázích vývoje. První fází byl vývoj inicializačního prototypu, který obsahoval všechny základní funkce aplikace. Druhá fáze přinesla vylepšení ve směru práce uživatele v hlavním rozhraní aplikace. V třetí fázi došlo k zásadnímu rozšíření aplikace o nový uživatelský prvek, popisku v layoutu. Ve čtvrté fázi vývoje byly v aplikaci odlaďovány *user-friendly* prvky a aplikace byla lokalizována do českého jazyka a v konečné fázi testována pro reálné nasazení.

Po otestování finální verze byl vytvořen instalační balíček, který měl zajistit jednoduchou instalaci pomocí instalačního průvodce. Tento instalační balíček byl doplněn o podrobný návod k použití, který uživatele provede všemi úkony spojenými s používáním aplikace.

Nyní se projekt nachází ve fázi, kdy je aplikace nasazena v reálném provozu. Nicméně jako u každého takového projektu se počítá s dodatečnými úpravami, které budou přispívat ke zkvalitnění produktu a zvýšení komfortu práce s aplikací.

Práce na projektu byla zajímavou zkušeností, zejména v posledních fázích testování a přizpůsobování aplikace uživatelům. V této fázi byl zřetelný rozdíl ve vnímání použitelnosti grafického rozhraní z hlediska programátora a z hlediska běžného uživatele. Programátor vidí jako jasnou výhodu možnost komplexního nastavení aplikace a velké

nabídky voleb, naopak běžný uživatel se spokojí s méně obsáhlým rozhraním s více automatizovanou činností aplikace. Úpravy při dotváření rozhraní byly ulehčeny zejména vlastnostmi technologie WPF pro tvorbu uživatelského rozhraní, která díky oddělení grafické a programové části umožňovala jednoduše měnit podobu prvků a doplňovat funkcionality zpříjemňující práci uživatele.

Při vytváření aplikační logiky jedinečně posloužila komponenta LINQ, která jednoznačně ulehčila operace nad databází, ale i tvorbu rozsáhlých XML struktur. Urychlila tak práci a umožnila dokonaleji propracovat samotné operace. Svůj přínos měla komponenta i z hlediska bezpečnostního, kdy byla právě prostřednictvím LINQ omezena přímá interakce uživatele s databází na minimum a omezily se tak nechtěné nestandardní zásahy běžného uživatele.

Přínosem pro zadavatele je možné považovat, tak jako u všech aplikací podobného typu, časovou úsporu a zefektivnění pracovního procesu týkajícího se realizace dohledu nad výrobními halami. Celá původní činnost zahrnující práci nejméně dvou zaměstnanců se zredukovala na jednoho zaměstnance, který je schopen intuitivně vytvořit layout výrobní haly pro vizualizaci výroby a v případě potřeby tento layout upravit, znovu použít nebo odstranit z vizualizace. Momentálně se další rozsáhlejší úpravy aplikace neplánují, nicméně dále by mohla být rozšiřována úprava layoutu v tabulkovém rozhraní, rozsáhleji automatizováno vytvoření layoutu ve formě automatického rozmístění strojů na základě požadavků popřípadě propojení aplikace s vizualizačními aplikacemi a vytvoření jednotného funkčního celku.

Z výše uvedeného je patrné, že *stanovené cíle práce byly splněny.*

ZÁVĚR V ANGLIČTINĚ

As an object of diploma thesis, the desktop application was realized from its analysis of requirements and design of application to its complete implementation, testing and deployment for real use with all dependencies including user guide and installation package.

During the analysis of requirements, the collection of requirements was created on base of communication with submitter and after that the document was created, which summarized all the requirements. According to this document, functional and non-functional requirements were analyzed and in case of uncertainty the necessary details were specified. All requirements were clearly identified in package diagram of requirements and on base of diagram the use-case model were created, that should describe the interaction of system with user. According to this model, there were developed use-case scenarios that should clarify details of interaction with user.

After the steps above, the application design of visual and logical part was created. This exercise included design of each dialog, control and function related to application control and interface behavior.

The implementation itself consisted of four main phases of development. First phase included implementation of initialization prototype, which contained all basic functions of application. Second phase covered some improvements in way of work with main interface of application. In third phase, the application was extended with new user element in form of description in layout. The fourth phase of development included debugging of *user-friendly* elements and functions. The application was localized to Czech, and in last stage, it was tested for real use.

After the test of final version, the installation package was created, which should provide simple installation with installation wizard. The installation package was extended with detailed user guide, which shows all possibilities of application.

Today the project is at phase of use in real environment. However, like another similar projects, additional improvements and adjustments are expected that should contribute to quality of final product and increase comfort of work with application.

The work on this project was interesting experience, particularly at the final phase of testing and adapting the application for user. At this phase, there was obvious difference

between perception of graphical interface usability from programmer's and from common user's point of view. Programmer sees advantage in complex manual settings and large offer of options, in opposite the common user needs less complex interface and more automated operations in application. All modifications of user interface were simplified by features for creating rich user interface of WPF technology, that separates visual part of application from application logic and that enabled easily change the shape of controls and extend functions to make application more user-friendly.

By creating application logic, the LINQ component made big deal, it obviously made the work with database easier, even creating XML structures was simplified by this component. The component speeded up the work and made it possible to forge the logic of operations. One of the benefits of using LINQ was the security issue. Using the LINQ component the straight interaction with database was limited to minimum so the unwanted non-standard interventions of common user too.

The acquisition for submitter is, as in similar project, the saving of time (and money naturally) and more efficiency in process of production hall visualization realization. Originally the whole process included work of at least two employees, now it was reduced to one employee, who is intuitively able to create the layout for production hall visualization, modify, reuse or delete it from visualization if needed. Today bigger modifications are not planed, however more modifications could be done in table view, there could be more automated functions for creating layout according user defined requirements, and in further future, layout designer and applications for visualization could be merged in one product.

From the above mentioned it is obvious, that *all tasks of thesis were accomplished.*

SEZNAM POUŽITÉ LITERATURY

1. NIAN-SHING CHEN, S.Y. H. Applying Evolutionary Prototyping Model in Developing Stream-based Lecturing Systems. *Interactive Educational Multimedia*. 4. Barcelona (Španělsko): University of Barcelona, 2002.
2. MICROSOFT. MSDN. In: *Compiling MSIL to Native Code* [online]. 2012 [cit. 2012-Duben-14]. Dostupné z: [Compiling MSIL to Native Code](#)
3. MSDN Library..*NET Framework Conceptual Overview* [online]. 2012 [cit. 2012-Únor-19]. Dostupné z: <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>
4. GARRETT, J. J. *The Elements of User Experience, Second Edition: User-Centered Design for the Web and Beyond*. New Riders, 2010. ISBN 978-0-321-68865-1.
5. MSDN Library. *ntext, text, and image (Transact-SQL)* [online]. Dostupné také z: <http://msdn.microsoft.com/en-us/library/ms187993.aspx>
6. MICROSOFT. Frequently Asked Questions (LINQ to SQL). In: MICROSOFT. *MSDN* [online]. 2012, verze 2012 [cit. 2012-Duben-25]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb386929.aspx>
7. JOHNSON, Glenn. *MCTS Self-Paced Training Kit (Exam 70-516): Accessing Data with Microsoft® .NET Framework 4*. Redmond, Washington 98052-6399: Microsoft Press, 2011. ISBN 978-0-7356-6260-0.
8. STOECKER, Matthew. *MCTS self-paced training kit (Exam 70-511): Windows applications development with Microsoft .net Framework 4*. Redmond, WA: Microsoft Press, 2011. ISBN 978-073-5627-420.
9. MACDONALD, Matthew. *Pro WPF in C# 2010: Windows presentation foundation in .NET 4*. New York, N.Y.: Distributed to the book trade worldwide by Springer-Verlag, c2010, 1181 s. Expert's voice in .NET. ISBN 14-302-7205-8.
10. MICROSOFT. *MSDN Library* [online]. 2012 [cit. 2012-01-17]. Dostupné z: <http://msdn.microsoft.com/en-us/library>
11. SKEET, Jon. *C# in depth. 2nd ed. Stamford, CT: Manning, c2011, 554 s. ISBN 19-351-8247-1*.

12. DEITEL, Paul J, Harvey M DEITEL a Paul J DEITEL. *Visual C# 2010: how to program. 4th ed.* Upper Saddle River, N.J.: Pearson Prentice Hall, c2011, 959 s. *How to program series.* ISBN 01-321-5142-1.

CITOVANÁ LITERATURA

1. NIAN-SHING CHEN, S.Y. H. Applying Evolutionary Prototyping Model in Developing Stream-based Lecturing Systems. *Interactive Educational Multimedia*. 4. Barcelona (Španělsko): University of Barcelona, 2002.
2. MICROSOFT. MSDN. In: *Compiling MSIL to Native Code* [online]. 2012 [cit. 2012-Duben-14]. Dostupné z: Compiling MSIL to Native Code
3. MSDN Library..*NET Framework Conceptual Overview* [online]. 2012 [cit. 2012-Únor-19]. Dostupné z: <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>
4. GARRETT, J. J. *The Elements of User Experience, Second Edition: User-Centered Design for the Web and Beyond*. New Riders, 2010. ISBN 978-0-321-68865-1.
5. MSDN Library. *ntext, text, and image (Transact-SQL)* [online]. Dostupné také z: <http://msdn.microsoft.com/en-us/library/ms187993.aspx>
6. MICROSOFT. Frequently Asked Questions (LINQ to SQL). In: MICROSOFT. *MSDN* [online]. 2012, verze 2012 [cit. 2012-Duben-25]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb386929.aspx>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GUI – Graphical User Interface

CLR – Common Language Runtime

MS – Microsoft

XML – extensible markup language

LINQ – Language Integrated Query

WPF – Windows Presentation Foundation

XAML - Extensible Application Markup Language

LCD - Liquid Crystal Display

WCF - Windows Communication Foundation

JIT - Just in time

API - Application Programming Interface

SQL - Structured Query Language

TCL - Transaction Control Language

DDL - Database Design Language

CAD - Computer Aided Design

IO - Integritní omezení

SEZNAM OBRÁZKŮ

Obrázek 1: Ukázka monitorovacího zařízení.....	19
Obrázek 2: Ukázka stávajícího ručně tvořeného <i>layoutu</i>	19
Obrázek 3: Schéma systému <i>layoutů</i> v projektu Shop Floor Control.....	20
Obrázek 4: Package diagram funkčních požadavků	22
Obrázek 5: Package diagram nefunkčních požadavků	22
Obrázek 6: Use Case Model pro Layout Designer	23
Obrázek 7: Diagram <i>Evolučního prototypování</i> [zdroj vlastní].....	31
Obrázek 8: Příklad přístupu k databázi pomocí <i>SqlCommnad</i>	34
Obrázek 9: Příklad mapování třídy a dotaz nad databází	35
Obrázek 10: Ukázka vytvoření XML struktury.....	35
Obrázek 11: Výsledná struktura XML dokumentu.....	35
Obrázek 12: Návrh grafického zobrazení	38
Obrázek 13: Návrh tabulkového rozhraní.....	39
Obrázek 14: Návrh podoby dialogu pro práci s databází.....	40
Obrázek 15: Návrh dialogu pro editaci atributů objektu	40
Obrázek 16: Relační model databáze.....	43
Obrázek 17: Diagram návrhu funkcionality <i>Undo</i>	45
Obrázek 18: Ukázka inicializačního prototypu	47
Obrázek 19: Ukázka prototypu v prvním evolučním cyklu.....	48
Obrázek 20: Struktura databáze po rozšíření o <i>LayoutTexts</i>	49
Obrázek 21: Prototyp v druhém evolučním cyklu (funkce <i>ORTHO</i> a <i>SnapToGrid</i>).....	50
Obrázek 22: Prototyp v třetí evoluční fázi.....	51
Obrázek 23: Blokové schéma tříd aplikace Layout Designer.....	52
Obrázek 24: Rozložení prvků hlavního okna	53
Obrázek 25: XML struktura pro uložení <i>shape</i> v souboru <i>ShapeList.xml</i>	54
Obrázek 26: Definice zobrazení <i>shape</i> v <i>MainListBoxu</i>	55
Obrázek 27: Ukázka výběru prvků v <i>layoutu</i>	57
Obrázek 28: Ukázka kódu z event handleru <i>MainCanvasMouseLeftButtonDown</i>	58
Obrázek 29: Realizace výběru objektů v <i>layoutu</i>	59
Obrázek 30: Náhled tabulkového rozhraní	59
Obrázek 31: Použití filtru u <i>DataGrid</i>	60
Obrázek 32: Detail prvku pro zobrazení chyb	61

Obrázek 33: Definice podoby řádku <i>ErrorListBox</i> v XAML.....	61
Obrázek 34: Selekcce jmen při kontrole unikátnosti <i>WRKCTRID</i>	62
Obrázek 35: Implementace vyhledávání duplicit <i>WRKCTRID</i>	62
Obrázek 36: XML struktura pro uložení <i>layoutu</i> do souboru.....	64
Obrázek 37: Definice <i>Commands</i> u <i>ScrollViewer</i>	65
Obrázek 38: Dialog pro prohlížení databáze	66
Obrázek 39: Ukázky dialogů <i>LayoutMngr</i> , <i>ShapeMngr</i> a <i>LayoutItemMngr</i>	66
Obrázek 40: <i>PreferencesDialog</i> - dialog pro nastavení připojení k databázi	67
Obrázek 41: Implementace metody <i>Validate</i> třídy <i>LayoutClass</i>	68
Obrázek 42: Grafická podoba <i>LayoutItem</i> v <i>layoutu</i>	69
Obrázek 43: Grafická podoba <i>LayoutText</i> v <i>layoutu</i>	69
Obrázek 44: Formát záznamu v logovacím souboru	69
Obrázek 45: Implementace metody <i>Push</i> v třídě <i>LimitedStack</i>	70
Obrázek 46: Definice výplně pozadí <i>UserControl</i> prvku <i>SnapGrid</i>	71
Obrázek 47: Výsledné pozadí vytvořené pomocí mini-language.....	72
Obrázek 48: Struktura MSI package pro deployment aplikace	74

SEZNAM TABULEK

Tabulka 1: Přehled primárních případů užití a jejich identifikace.....	23
Tabulka 2: Tabulka hlavních komponent třídy <i>MainWindow</i>	54

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO DVD

PŘÍLOHA P II: NÁVOD NA ZPROVOZNĚNÍ APLIKACE

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO DVD

Zde je uvedena struktura adresářů přiloženého DVD:

- Database -obsahuje testovací databázi
- Install - obsahuje instalační balíček
- OfflineData - obsahuje data pro offline testování
- Source - obsahuje kompletní projekt pro Visual Studio 2010

PŘÍLOHA P II: NÁVOD NA ZPROVOZNĚNÍ APLIKACE

Aplikaci je možné používat i v offline režimu, v tomto případě ale nejsou přístupné funkce databáze.

INSTALACE PRO ONLINE TEST

1. Aplikaci nainstalujeme podle pokynů z adresáře Install na přiloženém DVD.
2. Pro běh aplikace v online režimu je nutné mít na počítači nainstalován Microsoft SQL Server 2008 R2.
3. Pro připojení testovací databáze provedeme tyto kroky
 - o Spustíme příkazový řádek *cmd* a zadáme příkaz:

```
sqlcmd -S Server\Instance
```

kde *Server* je hostname počítače a *Instance* je název instance serveru

1. Po připojení k serveru připojíme databázi k serveru pomocí tohoto příkazu:

```
USE [master]
GO
CREATE DATABASE Machine_Monitoring ON
( FILENAME = N'X:\Database\Machine_Monitoring.mdf' ),
( FILENAME = N'X:\Database\Machine_Monitoring_log.ldf' )
FOR ATTACH ;
GO
```

kde *master* je jméno master databáze na serveru a *X* je písmeno jednotky DVD. (podrobnější informace o připojení a odpojení databáze na <http://msdn.microsoft.com/en-us/library/ms165673.aspx>)

4. Pokud vše proběhlo v pořádku, je možné v aplikaci nastavit ConnectionString:

```
DataSource=Server\Instance;InitialCatalog=Machine_Monitoring;IntegratedSecurity=True
```

, kde *Server* je hostname počítače a *Instance* je název instance serveru

INSTALACE PRO OFFLINE TEST

1. Aplikaci nainstalujeme podle pokynů z adresáře Install na přiloženém DVD
2. Do složky *X:\Users\%UserName%\AppData\Roaming* nakopírujte obsah adresáře *OfflineData* z přiloženého DVD, kde *X* je jméno systémového disku a *%UserName%* je jméno vašeho účtu.
3. Při spuštění aplikace budete upozorněni na nedostupnost databáze a možnost načtení offline knihovny, kde zvolíte možnost *Ano*.