

Modulární řešení vizualizace výroby a kancelářského procesu na LCD panelech

Pavel Kučera

Bakalářská práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2012/2013

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Pavel KUČERA**
Osobní číslo: **A10071**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**
Forma studia: **kombinovaná**

Téma práce: **Modulární řešení vizualizace výroby a kancelářského procesu na průmyslových LCD panelech pro výrobu přístrojových transformátorů ABB Brno**

Zásady pro vypracování:

1. Zpracujte literární rešerši na téma tvorby klient-server aplikací pomocí platformy .NET.
2. Analyzujte uživatelské požadavky a navrhnete řešení.
3. Na základě analýzy vytvořte databázový model.
4. Vytvořte základní funkčnost pro server, uživatelské rozhraní ovládání a zobrazení dat.
5. Vytvořte uživatelskou dokumentaci.
6. Demonstrujte výsledky a provedte zhodnocení.

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. NAGEL, Christian. Professional C 4 and .Net 4. Indianapolis, IN: Wiley Pub., 2010, 1474 s. ISBN 04-705-0225-8.
2. SPAANJAARS, Imar. Beginning ASP.NET 4 in C and VB. Indianapolis, IN: Wiley Pub., 2010, XXXVI, 803 s. Wrox beginning guides. ISBN 04-705-0221-5.
3. EVJEN, Bill, Scott HANSELMAN a Devin RADER. Professional ASP.NET 4 in C and VB. Indianapolis, IN: Wiley Pub., 2010, IVIII, 1477 s. ISBN 04-705-0220-7.
4. DORRANS, Barry. Beginning ASP.NET security. Chichester, U.K.: Wiley, 2010, XX, 412 s. ISBN 978-047-0743-652.
5. WATSON, Ben. C 4.0: řešení praktických programátorských úloh. Vyd. 1. Brno: Zoner Press, 2010, 656 s. Encyklopedie Zoner Press. ISBN 978-80-7413-094-6.

Vedoucí bakalářské práce:

Ing. Erik Král

Ústav počítačových a komunikačních systémů


Datum zadání bakalářské práce:

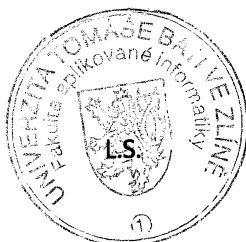
24. února 2013


Termín odevzdání bakalářské práce:

14. června 2013

Ve Zlíně dne 24. února 2013


prof. Ing. Vladimír Vašek, CSc.
děkan




prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Modulární řešení vizualizace výroby a kancelářského procesu na průmyslových LCD panelech pro výrobu přístrojových transformátorů ABB Brno.

Klíčová slova: Vizualizace, LCD, WPF, MVVM, MS SQL

ABSTRACT

A Modular Solution for the Visualisation, on Industrial LCD Panels, of Office and Plant Processes Regarding Instrument Transformer Production at ABB Brno.

Keywords: Visualisation, LCD, WPF, MVVM, MS SQL

Děkuji všem, kteří mě v průběhu studia podporovali, zejména rodině.

Prohlašuji, že

- beru na vědomí, že odevzdáním 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk 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 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 - 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 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....

podpis diplomanta

OBSAH

ÚVOD	12
I TEORETICKÁ ČÁST	12
1 MODEL KLIENT/SERVER	14
1.1 PROSTŘEDKY ARCHITEKTURY KLIENT-SERVER	14
1.1.1 PHP	14
1.1.2 My SQL	15
1.1.3 .NET Framework	16
1.1.4 MS SQL	16
1.2 KLIENT	16
1.2.1 Tlustý klient	16
1.2.2 Tenký klient	16
1.2.3 Windows Forms	17
1.2.4 Windows Presentation Foundation	17
1.2.5 Webová aplikace	17
1.3 SERVER	18
1.3.1 Web Services Description Language	18
1.3.2 Simple Object Access Protocol	18
1.3.3 REST	18
1.3.4 Windows Communication Foundation	19
2 DATABÁZE	20
2.1 NÁVRH DATABÁZE	20
2.2 NORMÁLNÍ FORMY	20
2.2.1 První normální forma	20
2.2.2 Druhá normální forma	20
2.2.3 Třetí normální forma	21
2.2.4 Boyce-Coddova normální forma	21
2.2.5 Čtvrtá normální forma	21
2.2.6 Pátá normální forma	21
3 LOKALIZACE APLIKACE WPF	22
3.1 NÁRODNÍ ZVYKLOSTI V .NET FRAMEWORK A WPF	22
3.2 ZDROJE PRO LOKALIZACI	22
3.2.1 Lokalizace založená na XAML	22
3.2.2 Lokalizace založená na RESX	23
4 NÁVRHOVÝ VZOR MVVM	24
4.1 VÝVOJ MVVM	24

4.2	PROČ JE VHODNÉ POUŽÍVAT NÁVRHOVÝ VZOR MVVM	25
4.3	VRSTVY	25
4.3.1	Model	25
4.3.2	View	25
4.3.3	ViewModel	25
4.4	PRINCIP	26
5	SILVERLIGHT	27
6	DATOVÝ PŘEVODNÍK	28
II	PRAKTICKÁ ČÁST ANALYTICKÁ	29
7	POŽADAVKY NA APLIKACI	30
7.1	POŽADAVKY Z POHLEDU LCD PANELU	30
7.1.1	Hardware	31
7.2	POŽADAVKY Z POHLEDU DOTYKOVÉHO LCD PANELU	31
7.3	POŽADAVKY NA AUTOMATICKOU AKTUALIZACI DAT	32
7.4	POŽADAVKY NA AKTUALIZACI SAMOTNÉ APLIKACE	32
7.5	APLIKACE OVLÁDÁNÍ	32
7.6	REALIZACE	32
7.6.1	AppSaver	33
7.6.2	AppUploader	33
7.6.3	ControlInterface	33
7.6.4	DataUpdaterMachine	33
7.6.5	LcdInterface	33
7.6.6	TouchLcdInterface	33
7.6.7	Server Service	33
7.6.8	Database	34
7.6.9	Přehled	34
III	PRAKTICKÁ ČÁST PROJEKTOVÁ	35
8	NÁVRH DATABÁZE	43
8.1	TABULKY PRO ULOŽENÍ SOUČÁSTÍ APLIKACE	43
8.2	CYKLY LCD PANELŮ	43
8.2.1	Ovládací PC pro zobrazení a rozvržení LCD panelu	44
8.2.2	Hlavní zobrazovací oblast a časová omezení pro zobrazení	45
8.2.3	Hlavní zobrazovací oblast a některé zobrazované prvky	45
8.3	NABÍDKY DOTYKOVÝCH LCD PANELŮ	46
8.4	AUTORIZACE UŽIVATELŮ DO APLIKACE OVLÁDÁNÍ	46
9	SLUŽBY SERVERU	51

9.1	WEB.CONFIG.....	51
9.2	POMOCNÁ TŘÍDA DBDATAEXCHANGE.CS	51
9.3	PŘÍKLAD SLUŽBY CONTROL INTERFACE SERVICE	52
9.3.1	Rozhraní služby IControlInterfaceService.cs	52
9.3.2	Služba ControlInterfaceService.svc.....	52
9.4	SLUŽBA FILE TRANSFER SERVICE	52
9.4.1	Rozhraní služby IFileTransferService.cs	52
9.4.2	Služba FileTransferService.svc.....	53
10	SPRÁVA VERZÍ APLIKACÍ	54
10.1	ULOŽENÍ	54
10.2	AKTUALIZACE	55
11	APLIKACE OVLÁDÁNÍ	57
11.1	MVVM V APLIKACI OVLÁDÁNÍ	57
11.1.1	RelayCommand.....	57
11.1.2	Hierarchie ViewModel	57
11.1.3	Třída ViewModelBase	58
11.1.4	Třída CommandViewModel	59
11.1.5	Třída MainWindowViewModel.....	59
11.1.6	Třída MainWindowWorkspacesVewModel	59
11.2	ORGANIZACE PROJEKTU.....	59
11.3	POPIS NABÍDEK	60
11.3.1	Přehled objektů pro zobrazení - obrázky	63
11.3.2	Rozvržení LCD panelů	65
11.3.3	Cykly LCD panelu.....	68
12	POMOCNÁ KNIHOVNA VISUALISATIONTOOLS	73
12.1	VÝBĚR DAT DLE ČASOVÉHO OMEZENÍ	73
12.2	TŘÍDY VIEWMODEL	73
13	APLIKACE ZOBRAZENÍ LCD	75
14	AUTOMATICKÁ AKTUALIZACE DAT	76
	ZÁVĚR.....	77
	SEZNAM POUŽITÉ LITERATURY	77
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	81
	SEZNAM OBRÁZKŮ	82
	SEZNAM TABULEK.....	83
	SEZNAM PŘÍLOH.....	84

ÚVOD

Téma této práce je "Modulární řešení vizualizace výroby a kancelářského procesu na průmyslových LCD panelech pro výrobu přístrojových transformátorů ABB Brno", analýza požadavků na aplikaci se tedy odvíjí od potřeb divize transformátorů ABB v Brně.

Produktivita této jednotky je rok od roku vyšší, čímž jsou kladeny čím dál větší požadavky na shromáždění, zpracování, vyhodnocení a vizualizaci dat z kanceláří a výroby. V kancelářském prostředí se sleduje například tzv. Office process, což je ukazatel, který pro každé oddělení (obchod, konstrukce, nákup,...) udává maximální dobu na odbavení probíhající zakázky. Ve výrobě lze sledovat a vyhodnocovat například postup rozpracované zakázky, rozpis změn, úspěšnost výroby nebo též zmetkovitost.

Již v minulých letech vznikl požadavek na vizualizaci těchto a dalších údajů na LCD panelech, aby bylo možné během několika okamžiků získat přehled o tom či onom výrobním nebo kancelářském úseku, nebo informovat zaměstnance.

Zdroj informací může být výstup z kteréhokoli nástroje kancelářských aplikací, může se nacházet na webových stránkách intranetu, může se jednat o určitý souhrn výrobních nebo procesních dat z databází (obvykle MS SQL) a v neposlední řadě může jít také o export ze SAP či jiných systémů. To vše takovým způsobem, aby informace nezastaraly, tedy je vyžadovaná aktuálnost zobrazených dat. Z tohoto bohatého výčtu je patrné, že komerční software tyto požadavky nedokáže splnit, nebo je velice nákladný.

Jako vizualizační jednotka se nabízel osobní počítač umístěný v blízkosti LCD panelu. Tato varianta se však ukázala jako nepřijatelná z důvodu vysoké ceny licencí na každém PC a také z důvodu agresivity prostředí, ve kterém se měl vyskytovat.

Všechny výše zmíněné indicie vedly k rozhodnutí vytvořit si vlastní softwarové a hardwarové řešení, které bude popsáno na následujících stránkách.

I. TEORETICKÁ ČÁST

1 MODEL KLIENT/SERVER

Informace jsem čerpal z databáze IBM: Software information center[1].

Běžným způsobem organizace softwaru distribuovaných systémů je rozdělení funkcionality na dvě části: klient a server. Klient je program, který využívá služby, které poskytuje jiný program. Program, který poskytuje služby se nazývá server. Klient vytváří požadavky na služby a server tyto služby provádí. Přitom funkce serveru vyžadují management zdrojů, ve kterém server synchronizuje a řídí přístup ke zdrojům, odpovídá na požadavky klienta daty nebo stavovými informacemi. Klientské programy typicky zajišťují uživatelskou interakci a dávají požadavky na data nebo iniciují změny dat na žádost uživatele.

Například Klientská aplikace poskytuje formulář, na kterém může uživatel zadat objednávky produktů. Klient pošle tyto informace o objednávce na server, který zkontroluje produkt v databázi a provede patřičné operace nutné pro zaplacení a zaslání zboží uživateli. Běžně je více klientských aplikací závislé na jednom serveru, přičemž serverové aplikace jsou připojené k jedné databázi.

Klient je izolovaný od řízení zdrojů serveru - nepotřebuje je znát. Jestliže je změněna databáze, kterou klient používá, může být nutná také změna serverové aplikace, ale klienta se tato změna netýká. Vzhledem k tomu, že serverová aplikace se skládá jen z několika málo částí, obvykle lokalizovaných na jednom aplikačním serveru, je taková aktualizace velmi jednoduchá. Tímto přístupem je také zajištěna větší bezpečnost, neboť pouze server, nikoli klient, přistupuje přímo do databáze.

Klient může také přistupovat k několika různým serverům a samotné servery mohou vystupovat jako klienti vůči jiným serverům. Přesný způsob, jak je funkcionalita distribuovaná napříč servery je záležitostí návrhu aplikačního řešení. Jeden server může například poskytovat veškeré služby, které klient vyžaduje a podle charakteru požadovaných dat nebo operací poté samotný server přistupuje na konkrétní části aplikace v rámci funkčnosti jiných serverů, nebo samotný klient může přímo dotazovat různé servery sám. Návrhář aplikace musí vzít v úvahu různé aspekty, jako je škálovatelnost, lokalizace a bezpečnost.

1.1 Prostředky architektury klient-server

1.1.1 PHP

Označení PHP bylo původně zkratkou anglické fráze "Personal Home Page". Tuto technologii vytvořil v roce 1994 Rasmus Lerdorf kvůli sledování návštěvníků svých stránek. S postupným nárůstem užitečnosti a možností této technologie (postupně se začala prosazovat ve stále profesionálnějších řešeních) se ujal název "PHP: Hypertext Preprocessor".[2]

PHP je skriptovací jazyk, který se provádí na straně serveru. Je vytvořený pro vývoj Webových aplikací, ale používá se také jako obecný programovací jazyk. V současné době je PHP nainstalováno na více než 244 miliónech webových stránkách a serverech. Momentálně je vytvářený a podporovaný PHP Group.[3]

Kód PHP je interpretovaný webovým serverem s modulem PHP preprocessor, který generuje výsledné stránky - příkazy PHP mohou být vloženy přímo do zdrojového kódu HTML dokumentů, což je výhodnější než volání externích souborů pro zpracování dat. Jazyk PHP se také rozvinul o rozhraní příkazové řádky a může být použit jako samostatná aplikace s grafickým uživatelským rozhráním.

PHP je svobodný software uvolněný pod licencí PHP License, která není kompatibilní s GNU GPL kvůli omezením na použití termínu PHP. PHP lze provozovat na většině webových serverů a také jako samostatnou aplikaci na téměř všech operačních systémech a platformách bezplatně.

1.1.2 My SQL

V tomto oddíle jsem čerpal z domovských stránek MySQL[4].

MySQL je nejoblíbenější a podle mnohých také nejlepší databázový systém s veřejným zdrojovým kódem. Ve skutečnosti, zejména když byly ve verzi 4 do systému přidány nové funkce, je MySQL životaschopným konkurentem drahých Goliášů, jakými jsou Oracle, nebo Microsoft SQL Server. Stejně jako jazyk PHP nabízí také systém MySQL výtečný výkon, přenositelnost a spolehlivost s přiměřenou dobou zaučení a minimálními náklady na provoz.[2]

My SQL je přibližně od roku 2008 nejvíce používaný open source relační databázový systém., který je zároveň serverovým systémem s víceuživatelským přístupem k databázím.

Projekt vývoje My SQL byl uvolněn pod licencí GNU GPL. Byl vlastněn a sponzorován švédskou firmou MySQL AB. Momentálně je vlastněn společností Oracle.

My SQL je populární volbou databáze pro uživatele webových aplikací a je centrální komponentou široce používaného svobodného softwaru LAMP (Linux, Apache, MySQL, Perl/PHP/Python), který zahrnuje výhody plnohodnotného databázového managementu.

Pro komerční použití nabízí většina placených edicí rozšířenou funkcionalitu. Aplikace, které používají databáze My SQL obsahují TYPO3, Joomla, WordPress, phpBB, MyBB, Drupal a další software. My SQL využívají také rozsáhlé webové systémy jako například Wikipedia, Google, Facebook, Twitter, Flickr, Nokia.com, a YouTube.

1.1.3 .NET Framework

.NET Framework, vyvíjený společností Microsoft je primárně určen pro Microsoft Windows. Obsahuje rozsáhlé množství knihoven, které poskytují potřebnou součinnost jazyků (každý jazyk může využívat kód napsaný v jiných jazycích). Programy napsané pro .NET Framework jsou prováděny v softwarovém prostředí Common Language Runtime (CLR), tedy na virtuálním stroji, který poskytuje řadu služeb, například zprávu paměti a zajištění spuštění kódu. Knihovna tříd a CLR společně tvoří .NET Framework. Bázové třídy .NET Framework poskytují uživatelské rozhraní, přístup k datům, konektivitu k databázím, kryptologii, vývoj webových aplikací, číselné algoritmy a síťové komunikace. Vývojář potom produkuje software, který je kombinací jeho vlastního zdrojového kódu, knihoven .NET Framework a ostatních knihoven. .NET Framework je používán většinou nových aplikací vytvořených pro platformy Windows. Microsoft také produkuje integrované vývojové prostředí pro .NET, Visual Studio.[5]

1.1.4 MS SQL

Microsoft SQL Server je relační databázový systém vyvíjený společností Microsoft. Primární funkcionalita databázového systému je ukládání a získávání dat v součinnosti s ostatními aplikacemi a to na stejném počítači nebo na jiném pomocí počítačové sítě (také Internetu). Existuje množství různých edicí Microsoft SQL Serveru, zaměřených na různé potřeby a zátěže (jestli databázi využívá jeden lokální počítač, nebo milióny uživatelů přes Internet ve stejném okamžiku). Dotazovacím jazykem je primárně T-SQL a ANSI SQL.[6]

1.2 Klient

1.2.1 Tlustý klient

Tlustý klient je počítač nebo aplikace nebo síť (klient) v architektuře klient-server, který poskytuje bohatou funkcionalitu nezávislou na centrálním serveru. Původně známý pouze jako klient je v protikladu s tenkým klientem, který je popisován jako velmi závislý na serverových aplikacích.

Tlustý klient stále potřebuje alespoň periodický kontakt se serverem, ale je charakteristický tím, že většinu funkčnosti dokáže zajistit bez připojení. Naproti tomu tenký klient provádí tak málo výpočtů, jak je to jen možné a závisí na přístupu na server pokaždé, když potřebuje zpracovat, nebo ověřit data.[7]

1.2.2 Tenký klient

Tenký klient je počítač (nejčastěji pouze počítačový terminál), nebo počítačový program, který je velmi závislý na jiném počítači (serveru), aby byl schopen zpracovat

výpočty nebo požadavky na data.[8]

1.2.3 Windows Forms

Windows forms je jméno grafického aplikačního programovacího rozhraní (Graphical Application Programming Interface - API), které je součástí Microsoft .NET Framework. Poskytuje přístup k nativním (přirozeným) prvkům rozhraní Microsoft Windows zapouzdřeným do řídicího kódu. Přestože je na Windows Forms nahlíženo jako na náhradu dřívějšího a více komplexního C++, založeného na Microsoft Foundation Class Library (MFC), neposkytuje náhradu návrhového vzoru Model-View-Controller. Pouze některé knihovny třetích stran poskytují tuto funkcionalitu.

Nejpoužívanější z návrhových vzorů, produkovaných firmou Microsoft je User Interface Process Application Block.[9]

1.2.4 Windows Presentation Foundation

Windows Presentation Foundation (WPF), vyvinutý společností Microsoft, je počítačový software - grafický subsystém sloužící k renderování uživatelského rozhraní v aplikacích založených na Windows. WPF, dříve známý jako Avalon, byl poprvé uvolněn jako součást .NET Framework 3.0. Raději než starší subsystém založený na GDI, WPF využívá DirectX. WPF se snaží poskytnout konzistentní programovací model na tvorbu aplikací a odděleného uživatelského rozhraní. Připomíná podobné objektové modely založené na XML, jako například XUL a SVG.

Jazykem WPF na tvorbu uživatelského rozhraní je XAML, což je jazyk založený na XML. Aplikace WPF mohou být samostatné desktop aplikace nebo objekty uvnitř webových stránek. WPF se zaměřuje na sjednocení běžných elementů uživatelského rozhraní, jako například 2D/3D renderování, adaptivní dokumenty, typografii, vektorovou grafiku, animaci a předrenderovaná média. Tyto elementy mohou být připojeny a manipulovány na základě různých událostí, uživatelské interakce a datových spojení.

Běhové knihovny WPF jsou začleněny ve všech verzích Microsoft Windows od verze Windows Vista a Windows Server 2008. Uživatelé Windows XP SP2/SP3 a Windows Server 2003 mohou volitelně doinstalovat příslušné knihovny.[10]

Microsoft Silverlight Microsoft Silverlight poskytuje funkcionalitu srovnatelnou s Adobe Flash a je součástí WPF. Jedná se například o běhový rendering ve 3D.[10]

1.2.5 Webová aplikace

Webová aplikace je taková aplikace, ke které uživatel přistupuje přes síť (intranet nebo Internet). V tomto smyslu se jedná o softwarovou aplikaci vytvořenou pro webové pro-

hlížeče v podporovaných jazycích (například JavaScript kombinovaný s renderovacím jazykem HTML) a jsou tedy závislé na prohlížeči, jak je bude vykreslovat.

Webové aplikace jsou populární kvůli všudypřítomnosti prohlížečů a výhodě používání webových prohlížečů jako klientů, někdy také zvaných tenký klient. Schopnost aktualizace a úprav aplikací bez nutnosti distribuce a instalace softwaru na tisíce potenciálních klientských počítačů je klíčovým důvodem jejich popularity, stejně tak jako jejich víceplatformní podpora. Běžné webové aplikace zahrnují například email, internetové obchody, aukce, encyklopedie...[11]

1.3 Server

Webová služba je metoda komunikace mezi dvěma elektronickými zařízeními přes WWW. Webová služba je softwarová funkce poskytující síťovou adresu pro přístup přes web. Tato služba je stále zapnutá.

W3C definuje webovou službu jako softwarový systém, navržený pro podporu interoperability interakce mezi zařízeními přes síť. Má rozhraní popsané ve strojově proveditelném formátu (WSDL). Jiné systémy komunikují s Webovou službou v předepsaným způsobem pomocí výměny zpráv SOAP, typicky za použití HTTP s XML serializací v součinnosti s ostatními webovými standardy.[12]

1.3.1 Web Services Description Language

WSDL je jazyk rozhraní založený na XML, který se používá k popisu funkcionality nabízené webovými službami. Popis webových služeb WSDL poskytuje strojově čitelný popis, jakým způsobem může být služba volána, jaké parametry očekává a jaké datové struktury vrací. Nabízí tedy účel, který úzce koresponduje s metodou podepisování v programovacím jazyce.

1.3.2 Simple Object Access Protocol

SOAP je protokol, který specifikuje strukturované informace implementované ve webových službách na počítačových sítích. Je založený na informační bázi XML popisující formát zprávy a obvykle je provozován na aplikační vrstvě. Je úzce spojen s HTTP nebo SMTP a zajišťuje spojení a přenos zpráv.

1.3.3 REST

Representative State Transfer (REST) je styl architektury softwaru velkých rozsahů, který poskytuje výhody technologií a protokolu WWW. REST popisuje, jak mohou být definovány a adresovány distribuované datové objekty pomocí škálovatelnosti a jednoduché výměny informací.[13]

1.3.4 Windows Communication Foundation

WCF, dříve známý jako Indigo je sada běhových programovacích rozhraní (application programming interface - API) v .NET Framework pro vytvoření spojení aplikací orientovaných na službu.

WCF je nástroj, často používaný k implementaci a vývoji architektur orientovaných na webové služby. Je vytvořený na těchto principech, aby podporoval distribuované výpočty tam, kde mají služby vzdálené klienty. Klienti mohou využít množství služeb; služby mohou být využívány mnoha klienty. Služby mohou být volně párovány jedna se druhou. Typicky mají rozhraní WSDL, které klient může využít k přístupu nezávisle na platformě, na které je služba spuštěna.

WCF implementuje mnoho rozšířených standardů webových služeb, například WS-Addressing, WS-ReliableMessaging a WS-Security. S uvolněním .NET Framework 4.0 poskytuje WCF také synchronizační službu RSS, WS-Discovery, routing a lepší podporu služeb REST.[14]

2 DATABÁZE

Databáze je organizovaná kolekce dat. Data jsou organizovaná dle návrhu nebo též modelu databáze, který vychází z analýzy potřeb aplikace, pro kterou je databáze určena. Při návrhu databáze je třeba vycházet z pravidel návrhu databáze. Samotný databázový server, na kterém je databáze spuštěná, musí mít dostatečný výkon, aby byl schopen obsloužit požadavky na zpracování dat jednoho a více klientů současně.

2.1 Návrh databáze

Návrh databáze je proces tvorby detailního datového modelu. Termín návrh databáze může být použitý k popisu mnoha různých částí celku, databázového systému. Principiálně, a nejspíše, se mluví o návrhu databáze jako o logickém designu datových struktur potřebných k uložení dat. V relačním modelu jsou to tabulky a pohledy. V objektové databázi entity a relační mapy vztahující se ke třídám a pojmenovaným vztahům. Často se ovšem návrhem databáze myslí také celkový proces vývoje databáze, tedy ne pouze datových struktur, ale také formulářů a dotazů použitých jako součást celku databázové aplikace systému řízení báze dat (Database Management System - DBMS).[16]

2.2 Normální formy

Při návrhu databáze jsou normalizační pravidla velmi důležitým krokem, Existují normální formy, které definují požadavky na vlastnosti schématu tabulky z pohledu závislostí mezi atributy.[15][17][18]

2.2.1 První normální forma

Tabulka je v první normální formě, jestliže všechny její atributy jsou atomické, to znamená, že jsou dále nedělitelné.

Každý atribut schématu relace je elementárního typu a je nestrukturovaný. Výskyt neatomických položek je obvykle řešen oddělením složených nebo opakujících se položek do nových relací.

První normální forma je jedinou normální formou, která je kriticky důležitá při návrhu a tvorbě vhodného designu tabulek. Je to základní podmínka "plochosti databáze". To znamená, že tabulka je dvourozměrné pole.

2.2.2 Druhá normální forma

Druhá normální forma se týká pouze tabulek, kde je volen primární klíč z více než jedné položky.

Tabulka je ve druhé normální formě, jestliže splňuje první normální formu a všechny neklíčové položky jsou závislé na celém (složeném) primárním klíči.

2.2.3 Třetí normální forma

Tabulka je ve třetí normální formě, je-li ve druhé normální formě a všechny atributy jsou přímo závislé pouze na primárním klíči. To znamená, že všechny neklíčové atributy jsou navzájem nezávislé.

2.2.4 Boyce-Coddova normální forma

Boyce-Coddova normální forma je považovaná za variaci třetí normální formy. Tabulka se nachází v Boyce-Coddově normální formě, jestliže pro každou netriviální závislost A na B platí, že A je nadmnožinou nějakého klíče schématu R . V podstatě tedy říká, že mezi kandidátními klíči nesmí existovat žádná funkční závislost.

2.2.5 Čtvrtá normální forma

Jedné hodnotě atributu odpovídá množina hodnot jiného atributu. Schéma relace R je ve čtvrté normální formě, jestliže pro každou netriviální multizávislost X platí, že X je nadmnožinou nějakého klíče schématu R .

2.2.6 Pátá normální forma

Pátá normální forma vychází z toho principu, že relaci lze dekomponovat na více tabulek a dekompozice musí být bezztrátová. Relace se tedy nachází v páté normální formě, pokud neobsahuje závislost podle spojení. Pokud závislost existuje, měla by být rozložena.

3 LOKALIZACE APLIKACE WPF

Lokalizace aplikace není pro každý scénář aplikace a jejího použití jednoduchý úkol. Proces je založen na několika klíčových principech, které jsou použity pro WPF a v obměnách také pro libovolné jiné typy klientských aplikací, které jsou v interakci s uživatelem. Je důležité, aby autor aplikace rozuměl základním regionálním zvyklostem zobrazení dat pro různé jazyky a kultury, ve kterých má aplikace komunikovat. Základní koncept je velmi podobný pro většinu klientských aplikací, ale konkrétní lokalizační proces pro WPF, tedy za použití jazyka XAML má určitá specifika.[19][20][21]

3.1 Národní zvyklosti v .NET Framework a WPF

Národní zvyklosti (informace a manipulace s nimi) jsou v .NET Framework přístupné pomocí instance třídy *CultureInfo*. Může být využívána jak ke zjištění stávajícího nastavení, tak také k nastavení specifické lokalizace aplikace. Každé vlákno obsahuje sadu informací o národních zvyklostech uloženou jako nastavení *CultureInfo* ve vlastnostech *CurrentCulture* a *CurrentUICulture*.

CurrentCulture definuje chování pro formátování a zobrazení (formát čísel, vzhled datumu, ...).

CurrentUICulture zajišťuje příslušné zdroje jazykové mutace daného vlákna.

Hlavní vlákno WPF bude mít vždy přiřazená nastavení *CurrentCulture* a *CurrentUICulture*. Tyto hodnoty je možné explicitně měnit v závislosti na požadavcích uživatele. Je velmi důležité si však uvědomit, že WPF spouští nová vlákna, která tato nastavení nemusí obsahovat. Ve skutečnosti nebudou nastavená vůbec, takže při explicitním nastavení jazyka je nutné počítat s nastavením všech spouštěných vláken.

3.2 Zdroje pro lokalizaci

Kromě již zmíněných nastavení zobrazení číselného nebo datumového formátu, je třeba také zajistit sadu názvů statických prvků formulářů (například názvy, zprávy, a podobně). Je třeba, aby tyto slovníky bylo možné snadno rozšířit o další jazyk, aby byly snadno přístupné a přehledné při vývoji aplikace a v neposlední řadě aby byly udržitelné při úpravách.

3.2.1 Lokalizace založená na XAML

Lokalizace založená na XAML využívá koncept XAML dokumentu a jeho specifické mapování individuálních prvků na vlastnosti elementů. XAML dokumenty v aplikaci poskytují základní mechanismus uložení zdrojů pro veškerý statický obsah, který je vytvořený v dokumentu XAML. Myšlenka je taková, že je v neutrálním jazyce vytvořený XAML dokument bez speciálního doplňkového kódu. Nástroj zvaný *LocBaml* lze

využít k exportu veškerého statického obsahu z kompilované aplikace do CSV formátu. Exportovaný zdroj může být přeložen a upraven do příslušného jazyka a následně opět pomocí LocBaml začleněn do aplikace.

Tato metoda je efektivní, ale statická. Uživatel tedy nemůže za běhu aplikace přepínat jazykové nastavení.

3.2.2 Lokalizace založená na RESX

Resx je tradiční a nejvíce podporovaná metoda .NET Framework. Silně typované Resx soubory zdrojů tvořené pomocí návrháře jsou dostupné přímo v kódu XAML. Existuje však mnoho způsobů, jakým je využívat a pro každý návrhový vzor je vhodný jeden konkrétní případ více než jiný. Stejně jako v případě BAML, je i zde možné nastavení ResourceFallback v případě nedostupnosti souboru jazyka.

4 NÁVRHOVÝ VZOR MVVM

Model-View-ViewModel je návrhový vzor pro aplikace WPF. Nabízí postupy a řešení, jak lze oddělit aplikační logiku od uživatelského rozhraní. Kódu je pak méně, je výrazně přehlednější, udržitelnější a případně změny nejsou tak problematické, obzvláště u rozsáhlejších aplikací.

MVVM oddělí data, stav aplikace a uživatelské rozhraní. Využívá se v něm binding a command - náhrada za uživatelské rozhraní řízené událostmi. Zdrojem pro následující informace je [24][26][27][28][29][22][30][31][32].

4.1 Vývoj MVVM

Již velmi dlouho existují různé návrhové vzory, které pomáhají návrhářům při tvorbě aplikací. Například návrhový vzor Model-View-Presenter (MVP) se těšil velké popularity při tvorbě uživatelských rozhraní na různých platformách. MVP je varianta jiného návrhového vzoru Model-View-Controller, který byl využíván po desetiletí. Návrhový vzor MVP je zjednodušeně popsán takto: To, co je zobrazeno je View, data, která jsou zobrazována se nazývají Model a Presenter je spojuje do jednoho celku. View je závislý na části Presenter, který jej plní daty, reaguje na uživatelský vstup, poskytuje vstupní kontrolu (obvykle delegací na funkcionality na Model), atd.

Martin Fowler publikoval článek o návrhovém vzoru Presentation Model (PM), který je podobný návrhovému vzoru MVP, ve kterém odděluje View od chování a stavů. Zajímavou částí návrhového vzoru PM je, že vytváří abstrakci pohledu zvanou Presentation Model. View se tedy potom stává pouhým nástrojem pro vykreslení, přičemž je často aktualizován pomocí tříd Presentation Model.

V roce 2005 představil na svém blogu John Gossman návrhový vzor Model-View-ViewModel (MVVM). MVVM je stejný s Fowlerovým Presentation Model v tom, že oba návrhové vzory poskytují abstrakci View, která obsahuje stav pohledu i jeho abstrakci. Fowler představuje Presentation Model ve smyslu tvorby uživatelského rozhraní, jako platformě nezávislou abstrakci uživatelského rozhraní, zatímco Gossman představuje MVVM jako standardizovaný způsob využití základních výhod WPF za účelem zjednodušení tvorby uživatelského rozhraní.

Oproti Presenteru v PVM, ViewModel nevyžaduje odkaz na pohled. Třídy View jsou vázány na vlastnosti tříd ViewModel, které naopak poskytují data obsažená v objektech model třídám View. Vazby mezi pohledy a třídami View model jsou jednoduše sestavitelné, protože ViewModel je nastavený jako datakontext pohledu. Jestliže se hodnoty vlastností ve ViewModel změní, tyto nové hodnoty jsou poskytnuty pohledu přes datové vazby. Když uživatel stiskne tlačítko v pohledu, je proveden příkaz ve ViewModel, který vybaví požadovanou akci. ViewModel zajišťuje modifikace dat, nikoli View.

Třídy pohledů nemají vůbec ponětí, že třídy model existují, dokud je třídy ViewModel nespojí a model se stejně tak nestará o pohled. Ve skutečnosti jsou třídy model známé pouze třídám ViewModel.

4.2 Proč je vhodné používat návrhový vzor MVVM

Kromě již zmíněných výhod, tedy přehlednost, udržitelnost a snadné úpravy zdrojového kódu, přináší MVVM celou řadu dalších. Velmi elegantním způsobem umožňuje oddělit práci na projektu pro Designéra od Programátora. Designér zpracuje návrh View, přičemž dodrží s programátorem domluvené rozhraní a oba mohou pracovat na své části aplikace vzájemně nezávisle.

ViewModel poskytuje veškerá data pro uživatelské rozhraní. Je velice podstatné, že svá data poskytuje v takových datových strukturách, které vyvolají události při jejich změně. To uživatelskému rozhraní umožňuje zobrazit automaticky nová data právě v okamžiku, kdy se ve ViewModel změní. ViewModel má dva základní prvky.

- `ObservableCollection<T>` hlásí, když je přidán nebo odebrán její prvek
- `INotifyPropertyChanged` popisuje události, které nastanou, když se změní některá vlastnost ViewModel

4.3 Vrstvy

4.3.1 Model

Popisuje data, se kterými aplikace pracuje. Zdrojem těchto dat mohou být databáze, datové soubory, a podobně. Model principiálně nesmí nic vědět o stavu ovládacích prvků.

4.3.2 View

View, tedy pohled reprezentuje uživatelská rozhraní, které je napsané v jazyce XAML. Může se jednat o formulář aplikace, webovou stránku, nebo ovládací prvek.

4.3.3 ViewModel

Třídy ViewModel spojují třídy Model a třídy View a zároveň udržují stav aplikace. Ovládací prvky (Views) jsou vázány (bindings) s ViewModel, z něhož čerpají obsah. Provádějí se v něm filtrace dat v závislosti na stavu aplikace. Aby mohl View poskytnout informace o změnách, implementuje rozhraní `INotifyPropertyChanged`. Totéž platí pro jeho vlastnosti typu kolekce.

4.4 Princip

MVVM vytvoří třídu, která drží stav aplikace, nazývá se ViewModel. Té se dotazuje uživatelské rozhraní, které vykresluje podle ní ovládací prvky. Naopak jestliže uživatel zadá do uživatelského rozhraní data, automaticky jsou aktualizována do ViewModel.

5 SILVERLIGHT

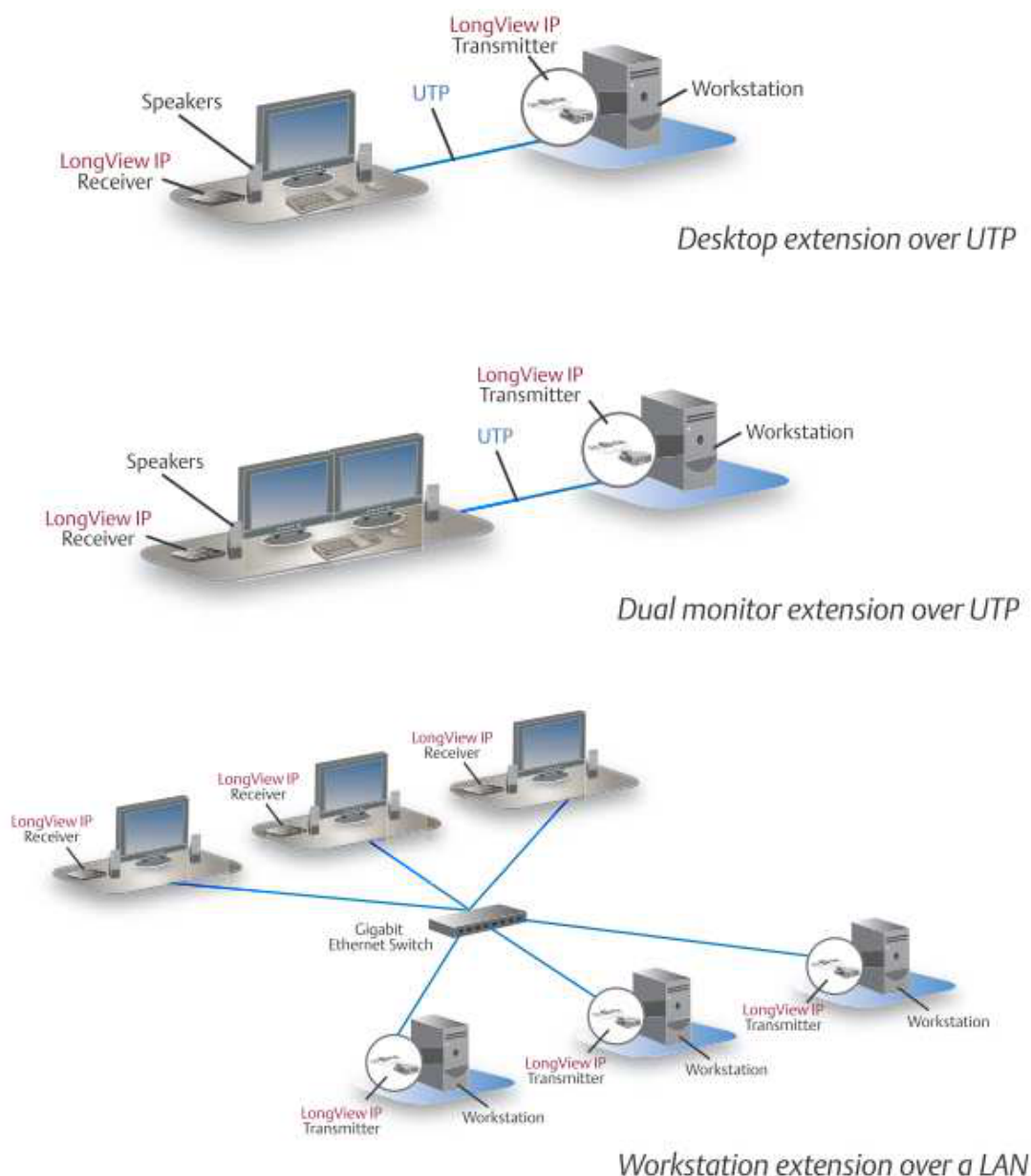
Silverlight je konstrukce na tvorbu aplikací pro webové prohlížeče, které jsou spouštěné na různých operačních systémech a poskytuje podobné možnosti jako Adobe Flash. Funguje díky pluginu doinstalovanému do prohlížeče. V okamžiku, kdy je na webové stránce obsažen prvek Silverlight, spustí se plugin a provede kód, čímž vyrenderuje obsah. Důležité je, že Silverlight poskytuje rozsáhlé prostředí, které je součástí HTML a JavaScriptu. Díky tomu lze na stránkách přehrávat video, poskytuje hardwarovou akceleraci 3D a dokáže zpracovat vektorovou animaci.[23][25]

Silverlight poskytuje také pro aplikace Windows integrovanou grafiku s pomocí WPF. Nabízí stejný rozsah funkcí jako pro Webové prohlížeče.

6 DATOVÝ PŘEVODNÍK

Datový převodník Avocent LongView IP je digitální zařízení pro zajištění vzdáleného přístupu k PC pomocí počítačové sítě (LAN). S pomocí LongView IP lze přistupovat ke všem standardním periferiím vzdáleného PC odkudkoli na počítačové síti. Poskytovaná rozhraní jsou USB, VGA/DVI, PS2 myš i klávesnice a sériový port.

LongView IP je tedy zařízení, které je primárně určeno pro vzdálený přístup k počítači ze vzdáleného terminálu, který může být pouze rozhraním uživatel-počítač. Příklady konfigurací jsou uvedené na obrázku obr. 1



Obr. 1. Možnosti zapojení datového převodník LondView IP[33]

II. PRAKTICKÁ ČÁST ANALYTICKÁ

7 POŽADAVKY NA APLIKACI

Cílem aplikace je zobrazení výrobních a procesních dat a informací pro pracovníky na průmyslových LCD panelech.

V rámci softwarové politiky ABB je třeba využívat podporované SW nástroje, což jsou v současnosti Microsoft Visual Studio a MS SQL Server 2008 R2. Aplikační a serverová část (tedy služby) má být psaná pomocí Visual Studia a MS SQL Server má být použitý pro uložení dat. Celkový přehled aplikací je uvedený na obrázku obr. 2.

7.1 Požadavky z pohledu LCD panelu

Požadavky na zdroj zobrazovaných dat se budou v průběhu života aplikace zcela jistě vyvíjet. Současné požadavky na zobrazovaná data jsou následující:

- Obrázky
- Prezentace PowerPoint
- Grafy a tabulky Excel
- Odkazy WWW (zde je třeba brát v úvahu, že některé stránky na firemním intranetu jsou přístupné po přihlášení)
- Videá
- Návštěvy (krátká informace o významných osobách firemní struktury, případně doplněná o fotografií)
- Standardní informace (v podstatě libovolný podporovaný formát zmíněný v tomto přehledu, ale zobrazovaný s jinou prioritou v zobrazovacím cyklu)
- Textové informace formou běžícího textu (počty kusů, stručné informace a podobně)
- Tabulky (data z databází firemní aplikační struktury, například výpis konkrétních dat z tabulek DB)
- Data ze SAP (z bezpečnostních důvodů je do SAP přístupováno například prostřednictvím Webové služby)

Cykly jednotlivých zobrazovaných prvků jsou řízené dle následujících scénářů:

- Typ zobrazení (celá obrazovka, menší rámeček, běžící text a jejich kombinace)
- Doba zobrazení každé informace

- Priorita opakování (některé informace, tzv. "Standardní informace" se zobrazují čistěji než v rámci celého cyklu jedenkrát)
- Videá a prezentace nejsou zobrazená pro přednastavenou dobu, ale od začátku do konce
- Jednotlivé prvky mohou být blokována, přeskakovaná, nebo naopak upřednostněné pro aktuální potřeby zobrazení
- Nastavení rozsahu datumů a časů pro zobrazení jednotlivých informací

7.1.1 Hardware

Na jednom PC s více výstupovou grafickou kartou je spuštěná jedna aplikace, která je kontejnerem pro formuláře jednotlivých LCD panelů. Tyto formuláře obsahují vlastní zobrazovaný obsah. V hlavní kontejnerové aplikaci je třeba umístit jednotlivé zobrazovací formuláře na správná místa dle připojení LCD panelů na výstupy grafické karty. Příklad je uveden na obrázku obr. 3.

Lcd panely jsou připojené ke grafické kartě prostřednictvím datových převodníků Avocent, které zajistí přenos obrazu prostřednictvím počítačové sítě. Tento přístup je zvolen z ekonomických důvodů. Není třeba pořizovat ke každému LCD panelu samostatné PC, čímž se ušetří zejména za náklady na licence instalovaného SW.

7.2 Požadavky z pohledu dotykového LCD panelu

Dotykový LCD panel by měl mít nabídku editovatelnou uživatelem, nebo administrátorem, který bude mít zodpovědnost za obsah. Obsahem nabídky, která tvoří strukturu (jejíž příklad je rozkreslený na obrázcích obr. 4 - Hlavní nabídka dotykového LCD panelu, obr. 5 - podnabídka Company dotykového LCD panelu, obr. 6 - podnabídka Products portfolio dotykového LCD panelu, obr. 7 - podnabídka Production dotykového LCD panelu a obr. 8 - podnabídka Other dotykového LCD panelu), potom budou dokumenty, které si může kdokoli zobrazit. Jedná se například o dokumenty PowerPoint, Videá, PDF a podobně.

Tento monitor není připojen ke společnému PC s jinými monitory, má svoje vlastní PC.

Důležitou vlastností aplikace dotykového LCD je, aby ji nemohl kdokoli vypnout a mít přístup k obsahu PC, neboť je na veřejně přístupném místě a zároveň jako přihlášené poskytuje přístup k firemním síťovým prostředkům a datům.

7.3 Požadavky na automatickou aktualizaci dat

Vzhledem k tomu, že ne všechny zobrazované informace mají svůj zdroj v aktivních databázích, může se jednat také o soubory MS Excel, které upravuje více uživatelů nezávisle na sobě prostřednictvím souborů spojených pomocí vzorců, je třeba zajistit automatickou aktualizaci těchto dat. To má na starost konzolová aplikace, která provádí import dat do databáze, odkud se potom budou zobrazovat na jednotlivé LCD panely. Zdrojem informace, které soubory je třeba aktualizovat je nastavení jednotlivých objektů, pomocí aplikace ovládání, při jejich zadání.

7.4 Požadavky na aktualizaci samotné aplikace

Jednotlivé aplikační a serverové součásti budou podléhat vývoji zejména v období nasazení aplikace do provozu a dále bude k drobným úpravám pravděpodobně docházet i v průběhu plného provozu aplikace. Vzhledem k tomu, že bude aplikace ovládání a aplikace zobrazení nainstalovaná na různých počítačích, není vhodné provádět aktualizaci aplikace po úpravách ručně, tedy obcházet uživatele. Za tímto účelem je třeba vytvořit vhodné nástroje, které budou zároveň odpovídat firemní bezpečnostní politice.

7.5 Aplikace Ovládání

Kromě již zmíněných funkcí LCD panelů, které musí aplikace ovládání zajišťovat, řízení přístupu k nastavení jednotlivých LCD panelů. Přístup bude mít povolená konkrétní skupina osob a k jednotlivým LCD panelům navíc podle přiděleného LCD panel, za který odpovídá. Administrátor aplikace má k dispozici navíc možnost přidávat samotné LCD panely.

Nastavení jednotlivých prvků bude možné přes příslušné formuláře. Nastavením se myslí jednak vložení obsahu do databáze a navíc taky vytvoření zobrazovacích cyklů obr. 9.

Ovládací aplikace musí obsahovat náhled zobrazení na každém LCD panelu takovým způsobem, jakým bude ve skutečnosti reálně fungovat.

7.6 Realizace

Jednotlivé prvky zmíněné v těchto kapitolách mají být zapojené do počítačové sítě. Nákres příkladu zapojení je uveden na obrázku 10. Součástí aplikace je AppSaver, AppUploader, ControlInterface, DataUpdaterMachine, LcdInterface, TouchLcdInterface, Server Service a Database.

7.6.1 AppSaver

Slouží k uploadu ControlInterface, DataUpdaterMachine, LcdInterface a TouchLcdInterface do tabulek v DB v průběhu testování a ladění aplikace a později také v případě úprav.

7.6.2 AppUploader

Stažení aktuální verze ControlInterface, DataUpdaterMachine, LcdInterface a TouchLcdInterface v případě, že je v DB nahraná aktuálnější verze.

7.6.3 ControlInterface

Uživatelské rozhraní, které slouží k nastavení LCD panelů: nahrání dokumentů (obrázků, tabulek, prezentací, videí, ...) do DB a nastavení pořadí, rychlosti, priority, ... , s jakou se mají zobrazit na jednotlivých LCD panelech. Každý panel má svoje vlastní nastavení.

ControlInterface bude napsaná ve WPF a C#, s využitím vzoru MVVM (Model View View-Model).

7.6.4 DataUpdaterMachine

Konzolová aplikace, která slouží k importu dat ze síťových disků ve firmě do DB, pro data, která je třeba pravidelně aktualizovat (například produkce).

7.6.5 LcdInterface

Rozhraní, které bude spuštěné na PC s více výstupovou grafickou kartou, bude zobrazovat data na jednotlivých LCD panelech podle toho, jak je uživatel nastaví pomocí ControlInterface.

Napsané v ve WPF a C#, s využitím vzoru MVVM.

7.6.6 TouchLcdInterface

Rozhraní pro dotykový LCD panel, bude zobrazovat uživatelem nastavenou nabídku, kterou může obsluha dotykového LCD procházet a spouštět vložené dokumenty, prezentace, videa atd.

Napsané v ve WPF a C#, s využitím vzoru MVVM.

7.6.7 Server Service

Webová služba WCF, umístěná na serveru, ke které budou přistupovat všechny výše zmíněné aplikace. Odděluje aplikace od databáze, provádí synchronizaci na LCD.

7.6.8 Database

Databáze MS SQL 2008 R2, ve které budou uložena data pro LCD panely a nové verze aplikací.

7.6.9 Přehled

V tabulce Tab. 1 je přehled jednotlivých zobrazovaných objektů, jaký je jejich zdroj (ve sloupci Ovládání), zda mají k dispozici automatickou aktualizaci, jakým způsobem jsou uloženy pro zobrazení na LCD, na které části LCD panelu dle obrázku obr. 3 mohou být zobrazeny, po jakou dobu a s jakou periodou opakování.

Tab. 1. Zobrazované typy dat

Promítaný objekt					
Ovládání	Zdroj pro aplikaci Automatická aktualizace	Zobrazení na LCD	Zobrazení	Doba zobrazení	Perioda zobrazení
Obrázky					
l/s adr.	NA	DB tbl Obrazky	A	Nast.	1x
Prezentace PPT					
l/s adr.	NA	DB tbl Prezentace	A	Dle PPT	1x
Grafy, tabulky XLS					
l/n adr.	Nast.	DB tbl Excel ...	A	Nast.	1x
WWW odkazy (pozor na přihlášení)					
Odkaz	NA	DB tbl Odkazy	A	Nast.	1x
Videa					
l/n adr.	NA	DB/Adr	A	Dle vid.	1x
Návštěvy					
Form.	NA	DB tbl Navstevy	B	Nast.	1xB
Standardní informace (jiná priorita cyklů)					
Dle typu	Dle typu	Dle typu, I	A	Nast.	Nx
Počty (kusů celkové, jednotlivé) pro běžící texty					
Dle typu T	Nast.	DB tbl Texty	B,C	Neust.	Řet.
Tabulky (uživatel zvolí zdroj a vytvoří pravidla pro vykreslení)					
DB SELECT	NA	DB tbl/v	A	Nast.	1x
Data ze SAP					
Web. služba	Nast.	DB Tmp	A,B,C	Nast	1x
Počasí					
Nast.Yahoo	NA	DB Tmp	B	Nast.B	1xB

Vysvětlivky zkratk uvedených v tabulce Tab. 1:

l/s adr — lokální nebo síťový adresář

Odkaz — platí pouze pro odkazy na WWW obsah, jedná se o http odkaz

Dle typu — platí pouze pro standardní informace. Standardní informací může být některý z ostatních objektů s tím, že má přiřazenou vyšší prioritu pro zobrazení. Zadáání se řídí pravidly příslušného objektu popsaného na ostatních řádcích.

Dle typu T — platí pouze pro běžící text. Může se jednat o ručně zadané informace, nebo vypočítávané hodnoty v databázi nebo například v buňce MS Excel.

DB SELECT — dotaz na data uložená v jiných SQL databázích

Web. služba — pro komunikaci se SAP

Nast. Yahoo — příklad pro získávání informací o počasí

NA — není k dispozici nebo nemá smysl

Nast. — dle nastavení v aplikaci ovládání

DB tbl... — umístěno v databázi LCD panelů

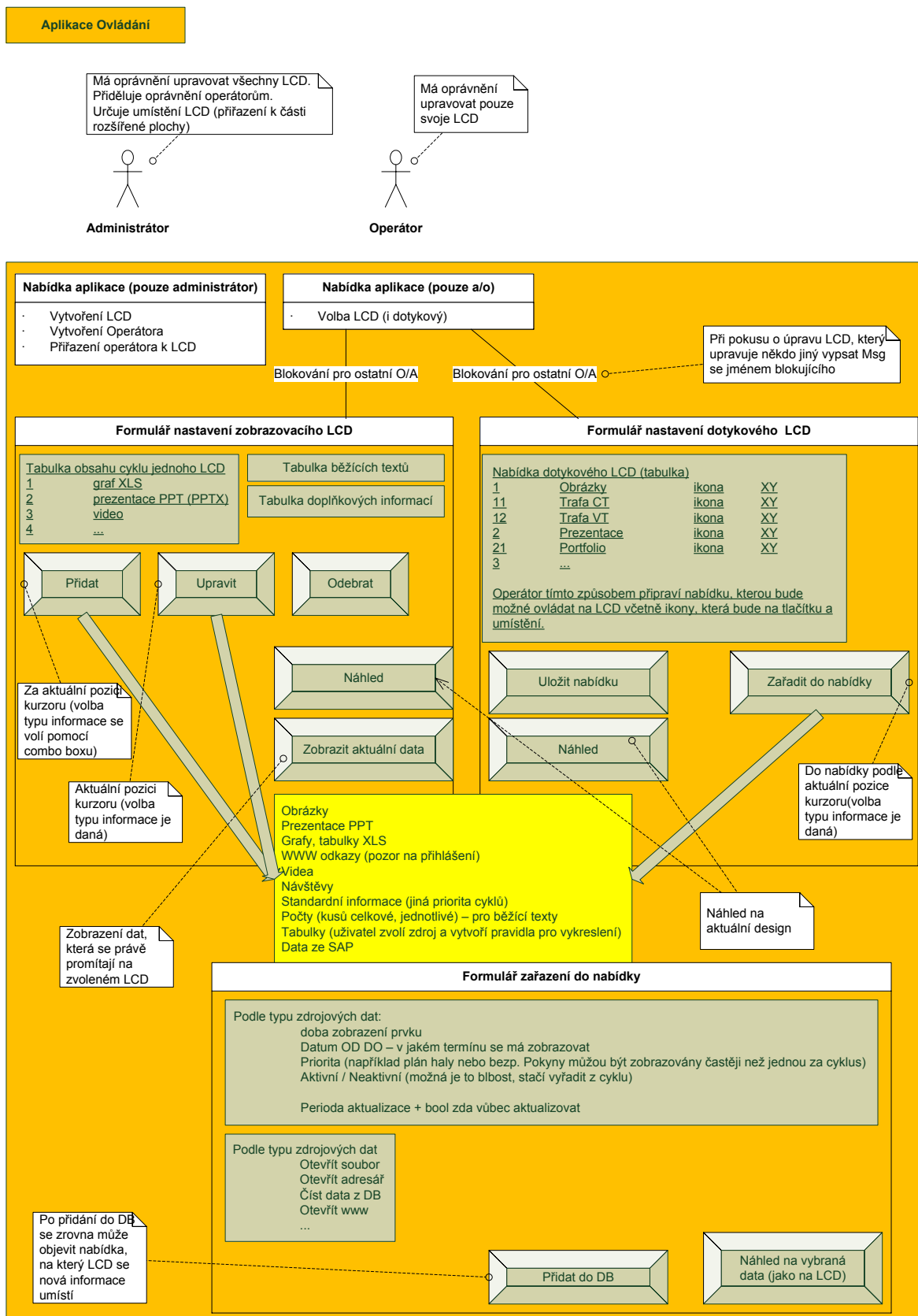
DB Tmp — dočasné tabulky v DB LCD panelů

Zobrazení A/B/C — umístění na LCD panelu dle obrázku obr. 3

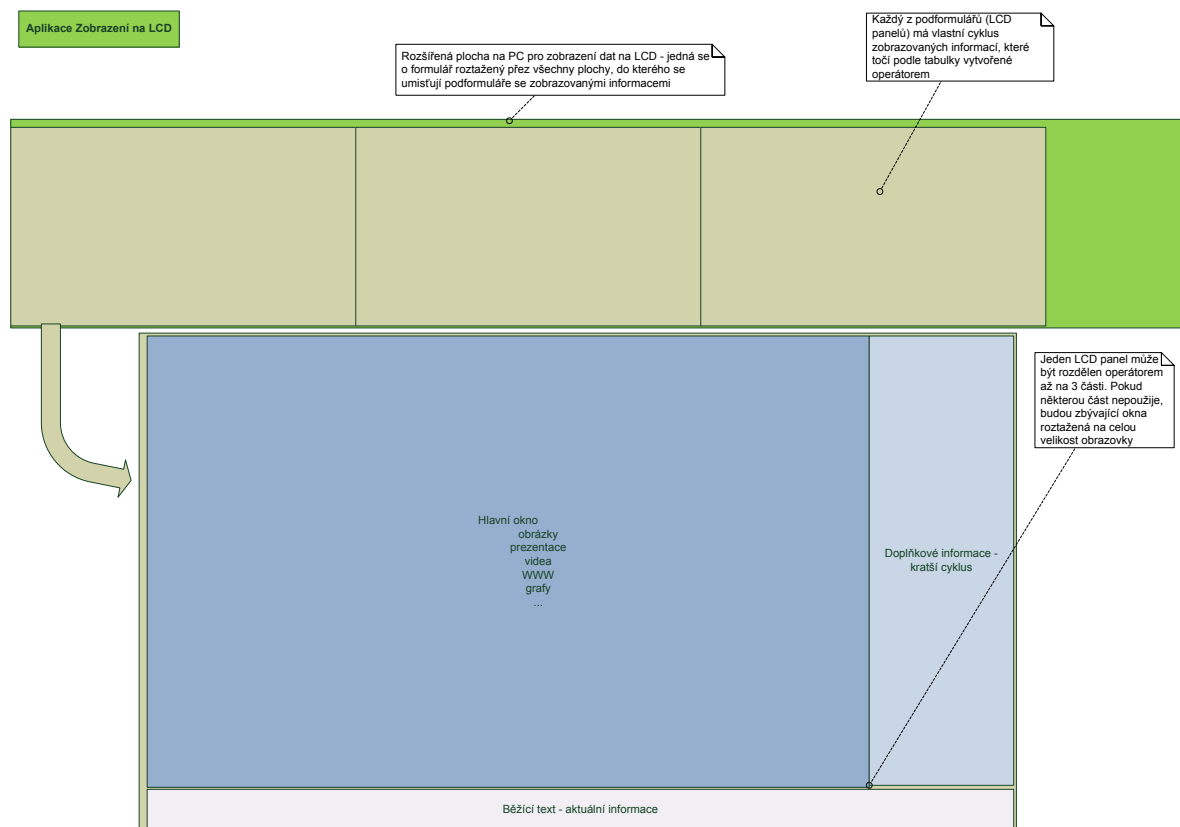
Dle PPT — doba zobrazení se nenastavuje, proběhne celá prezentace

Dle vid. — doba zobrazení se nenastavuje, promítne se celé video

Řet — pro běžící text se nenastavuje doba zobrazení, z požadovaných informací je sestavený řetězec, který neustále běží ve smyčce



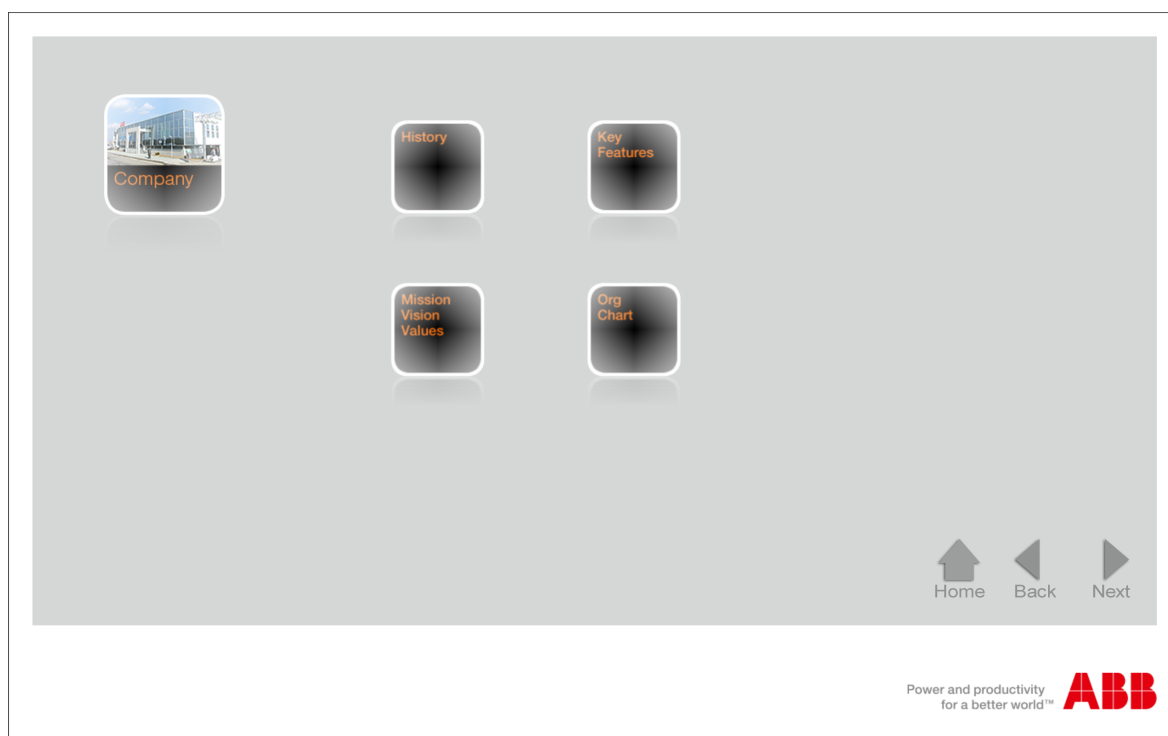
Obr. 2. Přehled funkcí aplikace ovládání LCD panelů



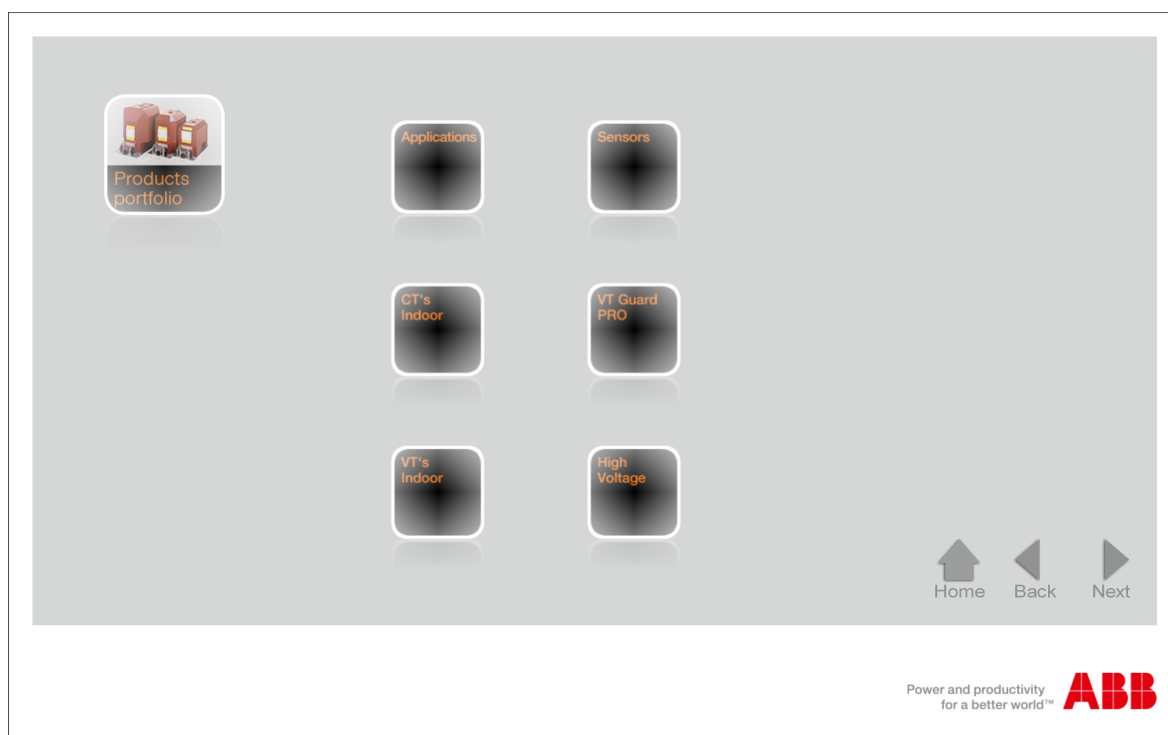
Obr. 3. Příklad LCD panelů připojených k PC



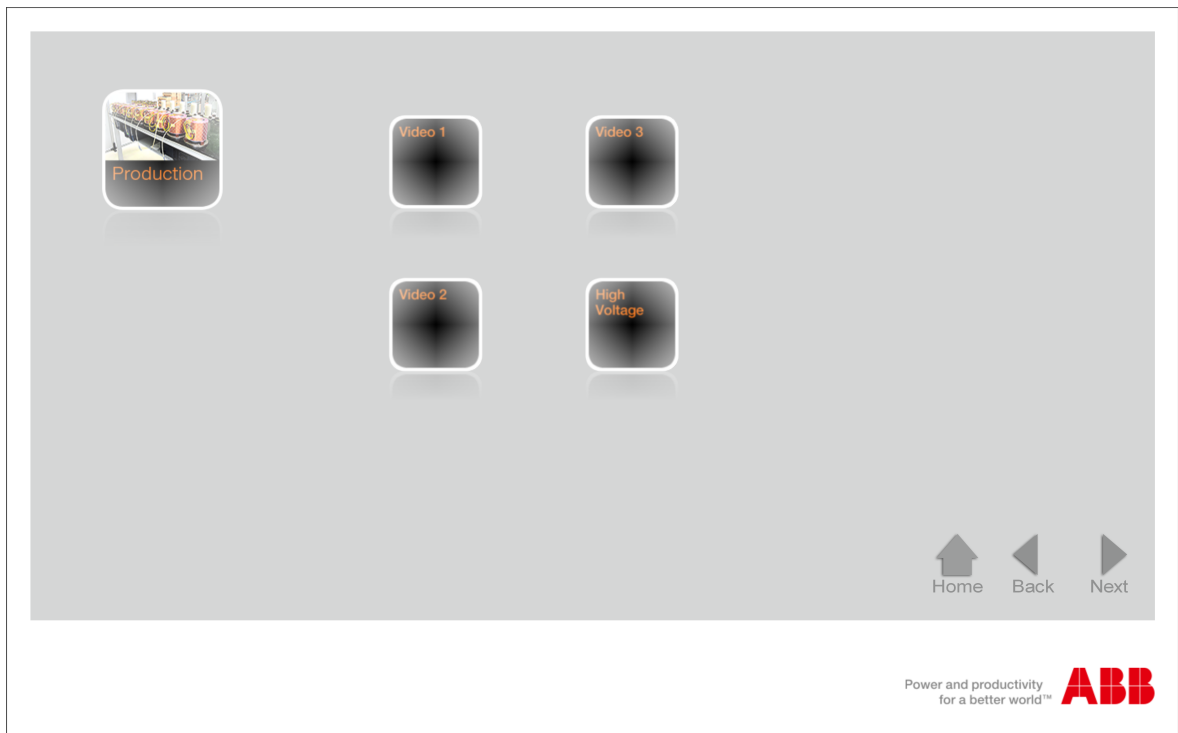
Obr. 4. Hlavní nabídka dotykového LCD



Obr. 5. Podnabídka Company dotykového LCD



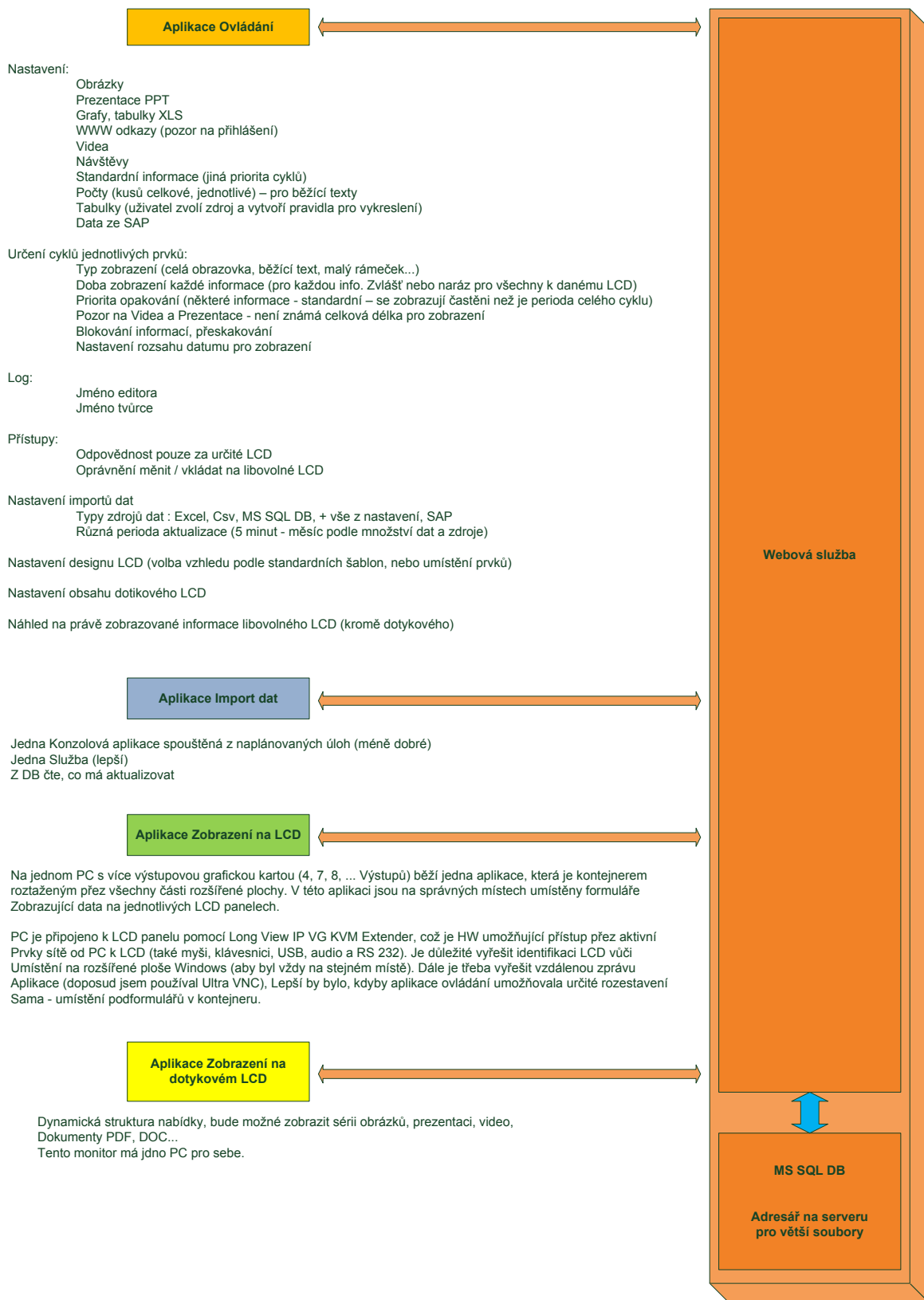
Obr. 6. Podnabídka Products portfolio dotykového LCD



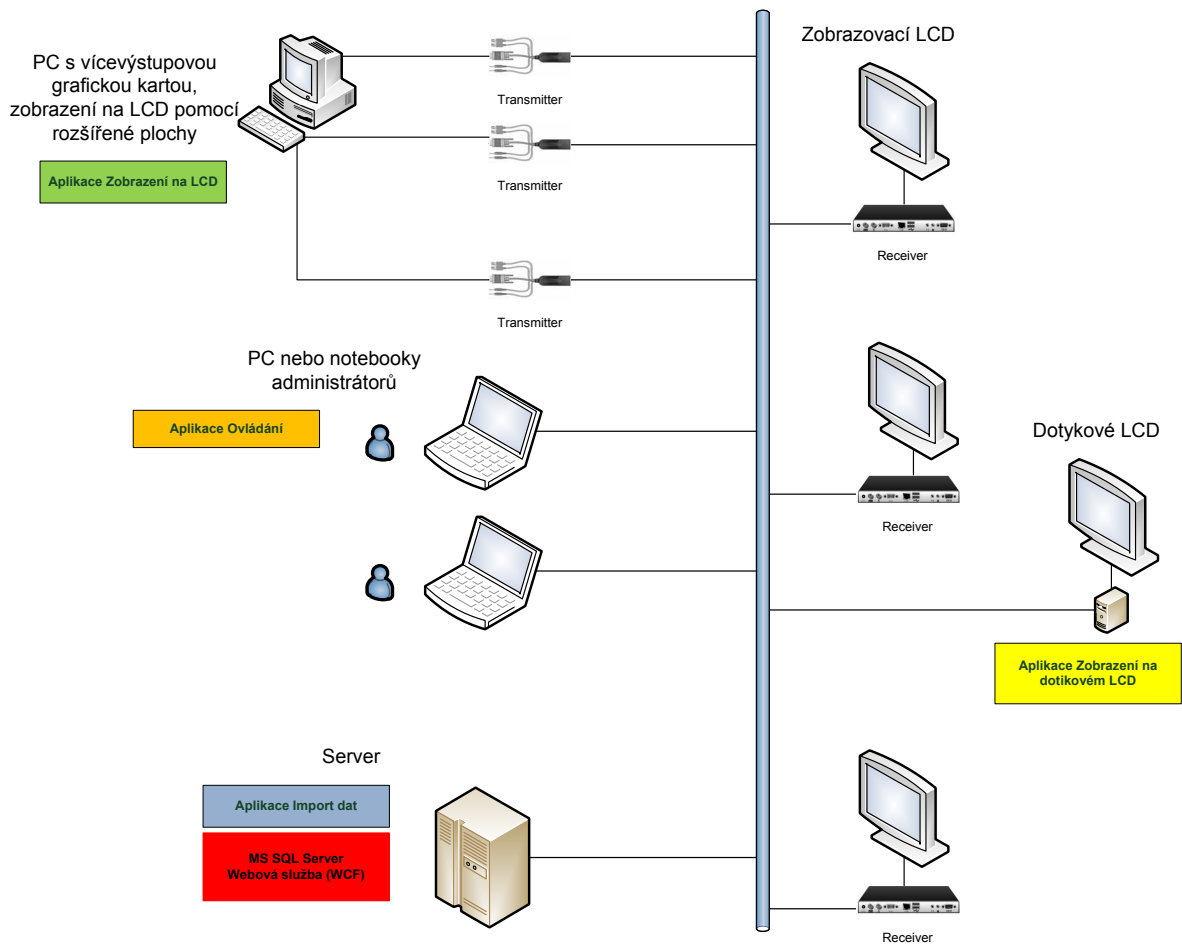
Obr. 7. Podnabídka Production dotykového LCD



Obr. 8. Podnabídka Other dotykového LCD



Obr. 9. Přehled funkcí aplikace ovládání LCD panelů



Obr. 10. Příklad struktury zařízení na síti

III. PRAKTICKÁ ČÁST PROJEKTOVÁ

8 NÁVRH DATABÁZE

Základním stavebním kamenem celé aplikace je databáze. Uložená data v databázi jsou jednak součástí aplikace, které slouží k automatické aktualizaci jednotlivých klientských i zobrazovacích součástí, dále informace o uživateli a jejich přístupech k jednotlivým nastavením, a v neposlední řadě také zobrazovaná data.

Používám jednu databázi, ve které jsou vytvořené tabulky související s jednotlivými součástmi aplikace. Na straně databáze probíhá editace dat prostřednictvím uložených procedur, případně prostřednictvím pohledů.

V následujících podkapitolách se budu zabývat strukturou databáze a popisem jednotlivých tabulek.

8.1 Tabulky pro uložení součástí aplikace

Součásti aplikace, které se ukládají s pomocí aplikace AppSaver a jsou automaticky aktualizovány na klienty pomocí aplikace AppUpdater jsou:

- Control interface
- Lcd interface
- Touch LCD interface
- Data update machine

Struktura tabulek, zobrazených na obrázku obr. 11 je pro všechny čtyři součásti stejná. Obsahuje primární klíč *ID*, dále jméno souboru *NameOfFile*, cestu souboru *PathOfFile*, která je uložena, aby bylo možné jednoduše ukládat aktuální release aplikace pomocí jednoho nastavení součástí, samotná data souboru *DataOfFile* a nakonec aktuální datum a čas aktualizace uloženého souboru *DateTimeOfFile*, který slouží pro automatickou aktualizaci, aby se z databáze stahovali pouze změněné soubory.

8.2 Cykly LCD panelů

Na obrázku obr. 12 je část databázového modelu, která slouží k uložení zobrazovacích cyklů na LCD panelech. Tento model lze rozdělit na několik logických částí podle významu.

- uložení informací o ovládacích počítačích a LCD panelech na obrázku obr. 13
- časová omezení pro zobrazení na obrázku obr. 14
- uložení dat pro zobrazení a návrh cyklu, jejichž příklad je na obrázku obr. 15

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
NameOfFile	nvarchar(100)	<input type="checkbox"/>
PathOfFile	nvarchar(1000)	<input type="checkbox"/>
DataOfFile	varbinary(MAX)	<input checked="" type="checkbox"/>
DateTimeOfFile	datetime	<input type="checkbox"/>

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
NameOfFile	nvarchar(100)	<input type="checkbox"/>
PathOfFile	nvarchar(1000)	<input type="checkbox"/>
DataOfFile	varbinary(MAX)	<input checked="" type="checkbox"/>
DateTimeOfFile	datetime	<input type="checkbox"/>

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
NameOfFile	nvarchar(100)	<input type="checkbox"/>
PathOfFile	nvarchar(1000)	<input type="checkbox"/>
DataOfFile	varbinary(MAX)	<input checked="" type="checkbox"/>
DateTimeOfFile	datetime	<input type="checkbox"/>

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
NameOfFile	nvarchar(100)	<input type="checkbox"/>
PathOfFile	nvarchar(1000)	<input type="checkbox"/>
DataOfFile	varbinary(MAX)	<input checked="" type="checkbox"/>
DateTimeOfFile	datetime	<input type="checkbox"/>

Obr. 11. Návrh DB : Tabulky pro uložení částí aplikace

8.2.1 Ovládací PC pro zobrazení a rozvržení LCD panelu

Ovládací PC je počítač s více výstupovou grafickou kartou, ke kterému jsou připojené, prostřednictvím digitálního převodníku LongView IP, LCD panely ve výrobních a kancelářských prostorách. V databázi (návrh tabulky je na obrázku obr. 13) je ovládací PC popsáno pouze svým jménem *ComputerName* a primárním klíčem *ID*, jehož prostřednictvím jsou k ovládacímu PC vázaná rozvržení LCD panelů.

Rozvržení LCD panelu je v databázi uloženo v tabulce *tbl_LCD_Layout* uvedené na obrázku obr. 13. Pro databázi je rozvržení LCD popsáno primárním klíčem *ID* a pro uživatele, z důvodu přehlednosti, je popsáno jménem *LcdName*. Jak již bylo zmíněno, k ovládacímu PC je vázané cizím klíčem *ComputerId*.

Ve sloupcích *ResolutionH* a *ResolutionV* je uloženo rozlišení monitoru a ve sloupcích *PositionH* a *PositionV* je uložena pozice formuláře LCD panelu vzhledem k celkovému rozlišení rozšířené plochy ovládacího PC.

Ve sloupci *IsTouchPanel* je uložena informace, zda se jedná o dotykový LCD panel. Dotykovým LCD panelem se budu zabývat v následující kapitole.

Každý LCD panel musí mít hlavní zobrazovací oblast, jejíž rozměr je uložený ve sloupcích *MainAreaResolutionH* a *MainAreaResolutionV*. V případě, že LCD panel obsahuje pouze hlavní zobrazovací oblast, odpovídají rozměry rozlišení LCD panelu. To je zajištěno programově při ukládání konfigurace LCD panelu z aplikace ovládání.

Informaci, zda má LCD panel ještě doplňkovou zobrazovací oblast, nese sloupec *HasAdditionalArea*. Jestliže je to tak, jsou vyplněné také sloupce *AdditionalAreaResolutionH* a *AdditionalAreaResolutionV*, které obsahují rozměr doplňkové oblasti. Totéž platí pro sloupec *HasRunningTextArea*, který určuje, zda má LCD panel běžící text a *RunningTextHeight*, ve kterém je uložena výška běžícího textu.

8.2.2 Hlavní zobrazovací oblast a časová omezení pro zobrazení

Tabulka, která popisuje hlavní zobrazovací oblast, tedy všechny prvky, které se v cyklu zobrazují, je na obrázku obr. 14. Jednotlivé prvky mohou být omezené na základě požadavků uživatele tak, aby se zobrazovali pouze v určitém období.

V tabulce *tbl_LCD_CycleDateTimeLimitation*, na obrázku obr. 14, která má primární klíč *ID*, vázaný do tabulky *tbl_LCD_MainAreaCycles* a jméno omezení *CyclesName* kvůli přehlednosti pro uživatele, jsou cizí klíče tří typů omezení podle dne v týdnu *tbl_LCD_CycleDays*, podle času *tbl_LCD_CycleTimes* a podle datumu *tbl_LCD_CycleDates*.

V případě, že není zvoleno žádné omezení, prvek je zobrazován v cyklech neustále. Jestliže však je cyklus nastaven, bude prvek zobrazován v závislosti na omezení s tím, že omezení lze kombinovat.

8.2.3 Hlavní zobrazovací oblast a některé zobrazované prvky

Pro přehlednost jsem neuvedl na obrázku obr. 15 všechny tabulky, ale pouze příklad, který charakterizuje způsob uložení také pro ostatní objekty.

Nejprve se budu věnovat tabulce *tbl_LCD_MainAreaCycles*, jejíž popis jsem v předchozí podkapitole vynechal. Má primární klíč *ID*. Pomocí cizího klíče *LcdLayoutId* je vázaná na tabulku *tbl_LCD_Layout*. Kromě dalších cizích klíčů *PictureId*, *PowerPointPresentationId*, *ExcelGraphId*, *ExcelTableId*, *WwwReferenceId*, *VideoId*, *VisitorId*, *TextInformationId*, *TableId* a *SapDataId*, které odkazují vždy na jeden z konkrétních objektů pro zobrazení. Dále pak je ve sloupci *IsSpecialInformation* informace, zda se jedná o významný objekt, který má při zobrazení přednost před ostatními a bude zobrazen častěji, ve sloupci *OrderNumber* je pořadí zobrazení v cyklu, díky čemuž je možné kdykoli měnit pořadí zobrazovaných objektů. Kromě již zmíněného sloupce *DateTimeLimitationId* je ve sloupci *VisibleSeconds* doba v sekundách, po kterou má být objekt zobrazený na LCD panelu.

První z tabulek, obsahující konkrétní zobrazovaný objekt je *tbl_LCD_Content_Pictures*. Obsahuje primární klíč *ID*, kterým je vázaná do tabulek *tbl_LCD_MainAreaCycles* a *tbl_LCD_AditionalAreaCycles*. Informace o obrázku jsou ve sloupcích *PictureName* - jméno obrázku, *PicturePath* - cesta souboru kvůli případné aktualizaci, *PictureData* - obsah souboru a *CreatedAt* - datum a čas poslední aktualizace obrázku.

V polích *AuthorId* je ID autora, který první vložil obrázek do databáze a *LastEditorId* obsahuje odkaz na posledního editora. Oba sloupce jsou cizí klíče odkazující do tabulky *tbl_App_Users*.

Tabulka *tbl_LCD_Content_PowerPointPresentation* má prakticky stejnou strukturu jako tabulka *tbl_LCD_Content_Pictures*, stejně tak jako tabulka *tbl_LCD_Content_Video*,

kteřá na obrázku obr. 15 není uvedená a tabulka *tbl_LCD_Content_ExcelWorkBook*, do které má být uložený soubor Excel a je součástí databázové struktury, pro uložení grafů a tabulek, s tabulkami *tbl_LCD_Content_ExcelWorkBook_Graph* a *tbl_LCD_Content_ExcelWorkBook_TableArea*, do nichž lze uložit graf ze sešitu Excel, nebo oblast tabulky Excel.

8.3 Nabídky dotykových LCD panelů

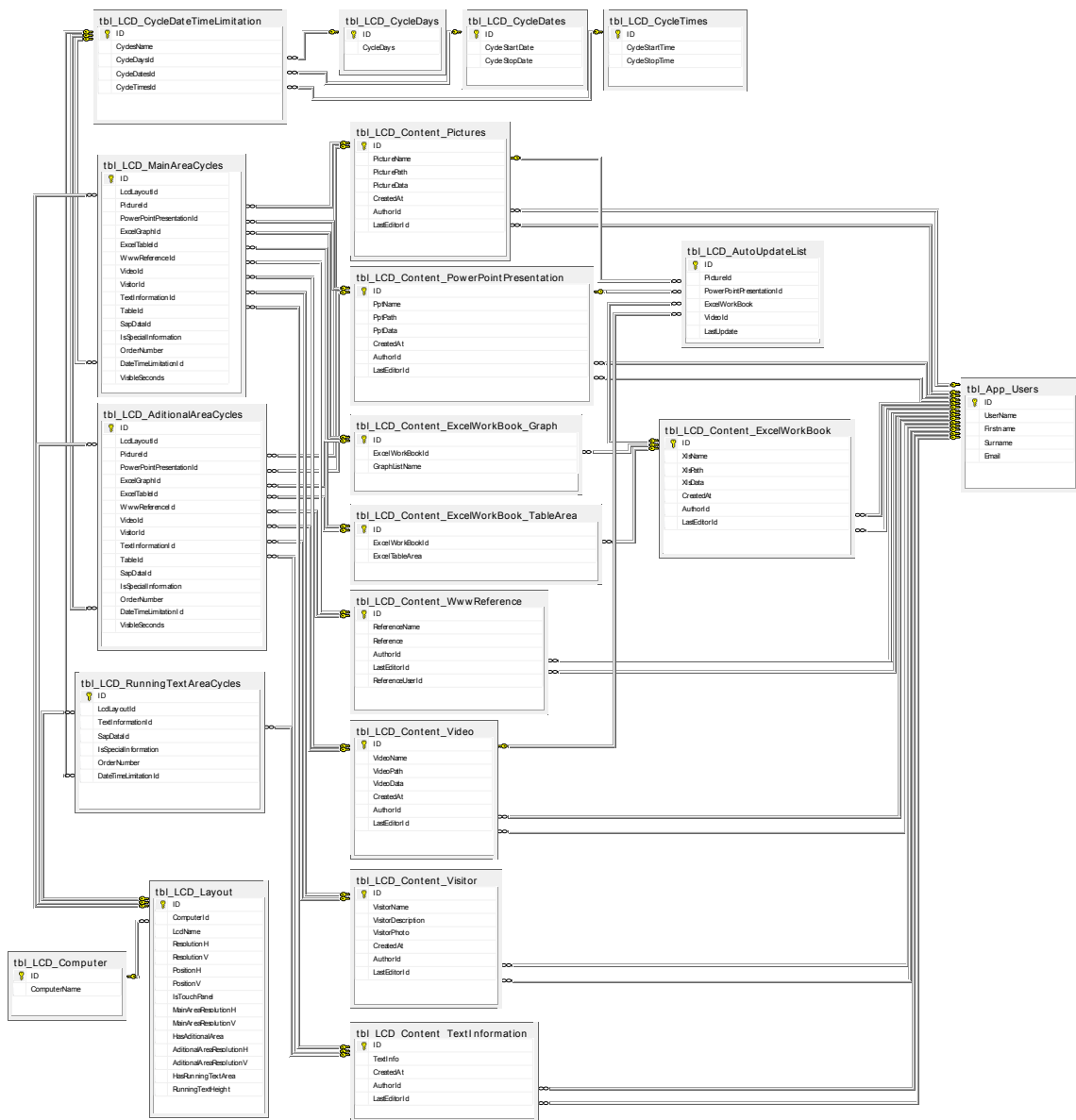
Na rozdíl od standardních LCD panelů, které zobrazují data cyklicky, dotykové LCD panely mají struktury nabídky, kterou je třeba procházet a data jsou zobrazovaná na žádost uživatele stojícího u LCD panelu. Databázový model pro uložení nabídek je na obrázku obr. 16. Je zde vidět, že *ID* tabulky *tbl_LCD_Layout* je cizím klíčem tabulky *tbl_LCD_TouchScreen*. Důvodem je, že jedna struktura nabídek může být určena pro více panelů.

Tabulka *tbl_LCD_TouchScreen* zároveň obsahuje uložené pozadí panelu. Design tabulky je vidět na obrázku obr. 17. Primární klíč této tabulky je cizím klíčem tabulky *tbl_LCD_TouchMenu*, která obsahuje prvky hlavní nabídky, tedy popis *Description*, data uložené ikony *IconText* a umístění ikony na panelu. Tabulka *tbl_LCD_TouchSubmenu* má stejnou strukturu s tím rozdílem, že je cizím klíčem *MenuId* vázaná k hlavní nabídce.

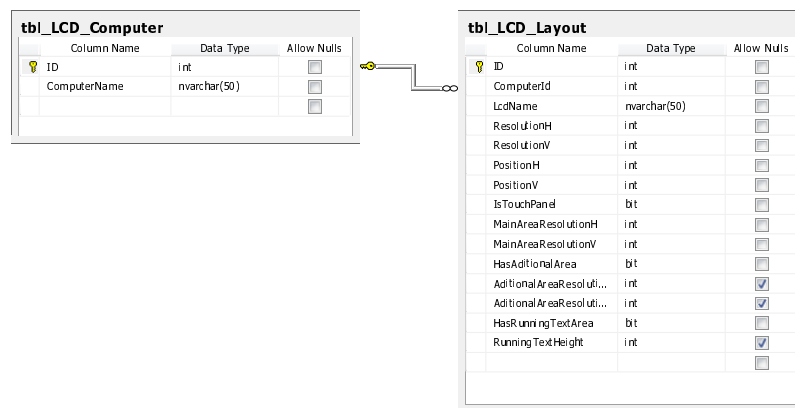
V tabulce *tbl_LCD_TouchList* jsou již cizí klíče (*PictureId*, *PowerPointPresentationId*, *WwwReferenceId*, *VideoId* a *DocumentId*), odkazy na zobrazované prvky.

8.4 Autorizace uživatelů do aplikace ovládání

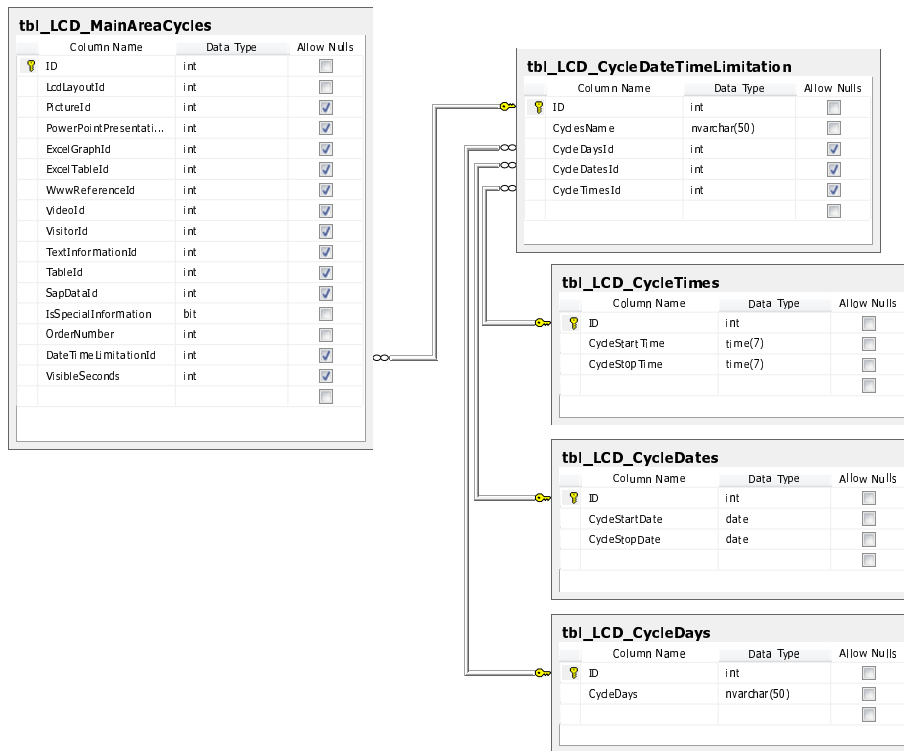
V modelu na obrázku obr. 18 jsou v tabulce *tbl_App_Autorization* uložena přístupová práva v podobě popisu jednotlivých prvků, ke kterým lze přistupovat, jednotlivých LCD panelů. Uživatelům jsou přidělována práva vazbou M:N pomocí tabulky *tbl_App_UsersAutorisation*.



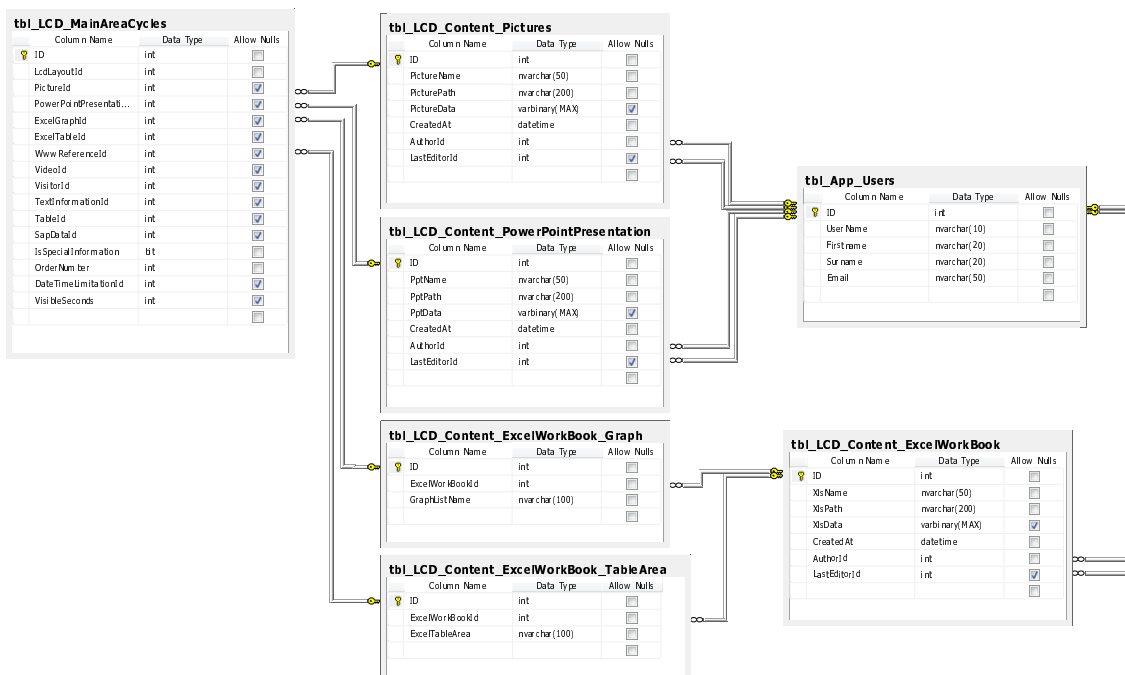
Obr. 12. Návrh DB : Celkový návrh cyklů LCD panelů



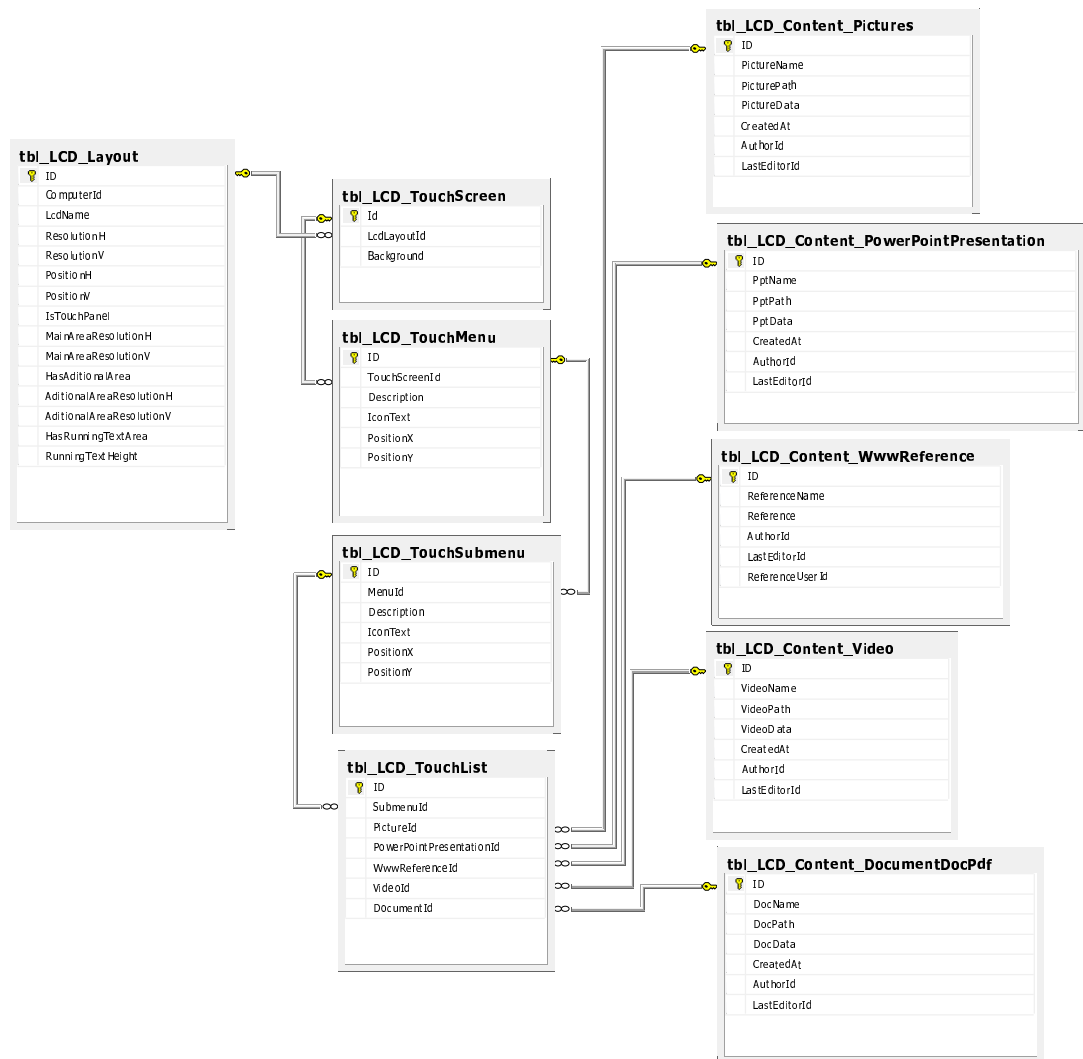
Obr. 13. Návrh DB : Cykly LCD panelů : Rozvržení LCD panelu a PC



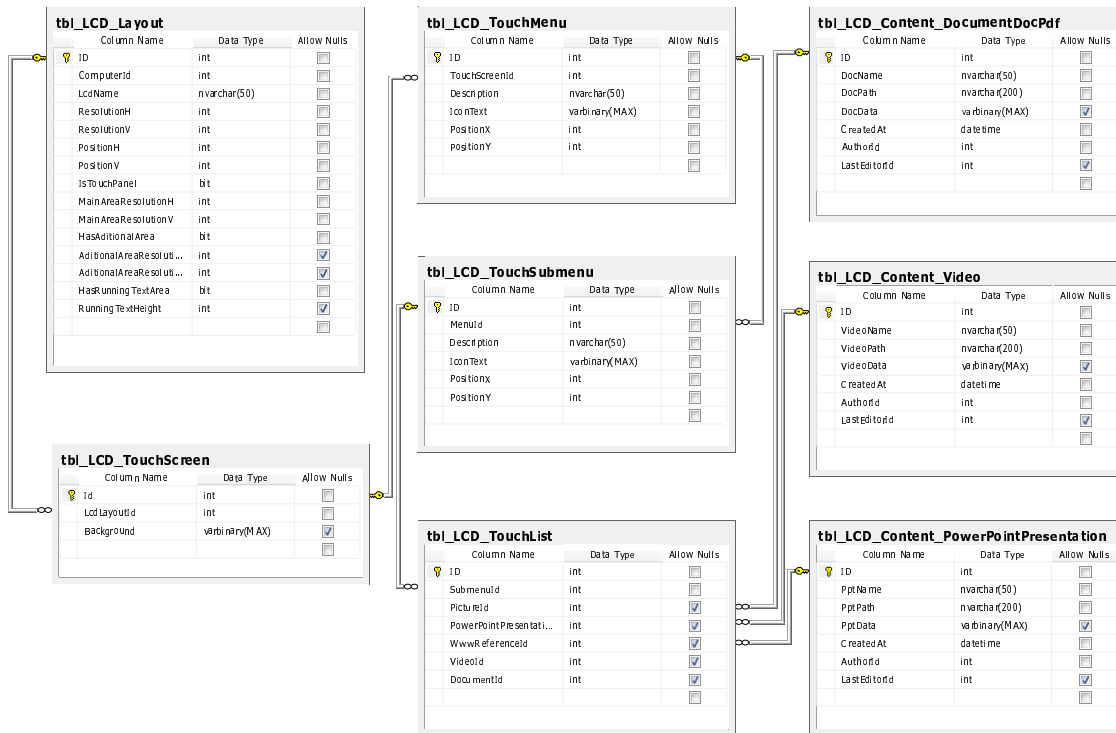
Obr. 14. Návrh DB : Cykly LCD panelů : Časová omezení cyklů



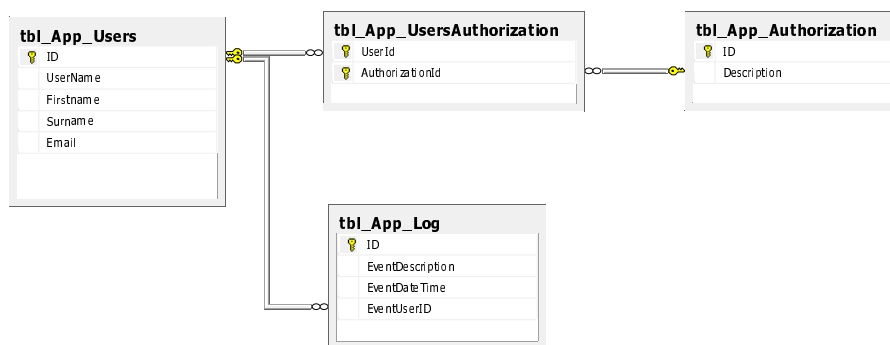
Obr. 15. Návrh DB : Detail části cyklů LCD panelů



Obr. 16. Návrh DB : Nabídky a obsah dotykových panelů



Obr. 17. Návrh DB : Detail nabídky a příklad obsahu dotykových panelů



Obr. 18. Návrh DB : Autorizace pro aplikaci ovládání

9 SLUŽBY SERVERU

Při tvorbě služeb WCF, jsem definoval operace pomocí metod a typů implementovaných s použitím .NET Framework. Zprávy, které jsou odesílány a přijímány webovou službou jsou definovány s použitím XML schématu. Běhová služba WCF konvertuje data na příchozí zprávy pro služby z tohoto XML formátu do typů .NET Framework a posílá je jako parametry metodám, které implementují operace služeb. Podobně když služba vrací hodnotu operace, běhová služba WCF konvertuje data zpět do XML a vysílá ji klientské aplikaci. Vzhledem k tomu, že jsem použil k tvorbě Visual Studio, je XML serializovaný proud dat konvertován opět do přirozeného jazyka .NET Framework dříve, než vstoupí do kódu klientské aplikace.

Při psaní služeb a jejich metod jsem bral v úvahu, že ne všechny kolekce jsou podporovány a prakticky v případě všech objektů posílaných prostřednictvím služeb jsem vytvořil odpovídající konvertor.

Pro komunikaci jsem zvolil *BasicHttpBinding*. Jedná se o základní komunikační spojení, využívající HTTP nebo HTTPS protokol. Vytváří spojení SOAP, založené na WSDL a XSD. Tato metoda je vhodná pro komunikaci tlustého klienta s webovou službou a pro moji aplikaci zcela dostačující.

9.1 Web.config

Velmi podstatnou součástí webové služby je správná konfigurace tohoto souboru. Pomocí tagů XML je zde definovaný typ komunikačního spojení, jeho kódování, maximální velikost příchozí a odchozí zprávy, timeout spojení, verze Framework a podobně.

Za povšimnutí ve výpisu uvedeném v příloze stojí speciálně konfigurovaná služba *FileTransferService*. Je určena k přenosu velkých objemů dat a proto jsou hodnoty komunikačních limitů nastavené na maximum.

9.2 Pomocná třída *DBdataExchange.cs*

Metody třídy *DBdataExchange.cs*, vypsané v příloze, slouží ke komunikaci služeb WCF s databází MS SQL.

V konstruktoru třídy jsou předávány pouze parametry *userID*, což je ID uživatele provádějícího na klientské aplikaci požadavek na server. Slouží k verifikaci oprávnění a zápisu do aplikačního loku (tabulky databáze) v případě, že se jedná o editaci záznamů. Druhým parametrem konstruktoru je název uložené procedury databáze MS SQL.

Komunikaci potom zajišťuje metoda *public DataSet DataExchange(DataTable dtParameters)*. Jestliže parametr *dtParameters* obsahuje nějaké řády, tedy je volaná procedura DB s parametry, je nejprve z databáze načten seznam parametrů a k nim jsou potom přiřazeny *dtparameters*.

Návratovým parametrem metody *DataExchange* je *DataSet*, jehož obsah je v kódu rozklíčovaný podle přiřazení k příslušným objektům.

Díky této třídě lze ušetřit kód na volání každé uložené procedury na tabulku parametrů a výpis návratových hodnot do seznamu.

9.3 Příklad služby *Control Interface Service*

9.3.1 Rozhraní služby *IControlInterfaceService.cs*

Jako příklad zde uvádím část rozhraní *IControlInterfaceService*, a to její část týkající se přenosu vizuálních objektů, tedy obrázků, prezentací, videí, atd. Jak je vidět, v rozhraní jsem definoval metody pro čtení seznamu objektů určitého typu *List<VisuObject> VisuObjectList(VisuObjectKind objectKind, int userID)*, uložení objektů *void VisuObjectsToSave(VisuObjectKind objectKind, List<VisuObject> visuObjects, int userID)*, výmaz objektů *void VisuObjectsToDelete(VisuObjectKind objectKind, List<VisuObject> visuObjects, int userID)* a uložení jednoho objektu *int SaveOneVisuObject(VisuObjectKind objectKind, VisuObject visuObject, int userID)*.

V části *Visualisation objects* jsou vyjmenované typy objektů *VisuObjectKind* a třída *VisuObject*, která je kontejnerem pro přenášený objekt typu obrázek, prezentace, video, soubor Excel a dokument Word nebo PDF. Ostatní objekty mají jinou strukturu.

9.3.2 Služba *ControlInterfaceService.svc*

Tato služba (jejíž část výpisu je uvedena v příloze), zajišťuje komunikaci s klientskou aplikací *Control Interface*. Má na starost konfigurační údaje v rámci typů a odvozených objektů, ale nemá na starost přenos velkých souborů.

Ve výpisu v příloze je patrné, jakým způsobem jsou mazány, zapisovány a aktualizovány vizualizační objekty obrázků a prezentace. Prakticky vždy se jedná o vhodné požití metod třídy *DBdataExchange* s příslušným požitím parametrů vstupních nebo výstupních objektů.

9.4 Služba *File Transfer Service*

Tato služba *IFileTransferService* slouží k přenosu velkoobjemových dat. Má na starost jak přenos souborů jednotlivých klientských aplikací kvůli aktualizaci, tak i vlastní vizualizační objekty.

9.4.1 Rozhraní služby *IFileTransferService.cs*

Z výpisu v příloze je patrné, že už rozhraní *IFileTransferService.cs* se od *IControlInterfaceService.cs* velmi podstatně liší. Místo přenášení dat pomocí parametrů a seznamů (resp. polí) objektů, využívá *Streamy* - tedy proudy dat.

9.4.2 Služba `FileTransferService.svc`

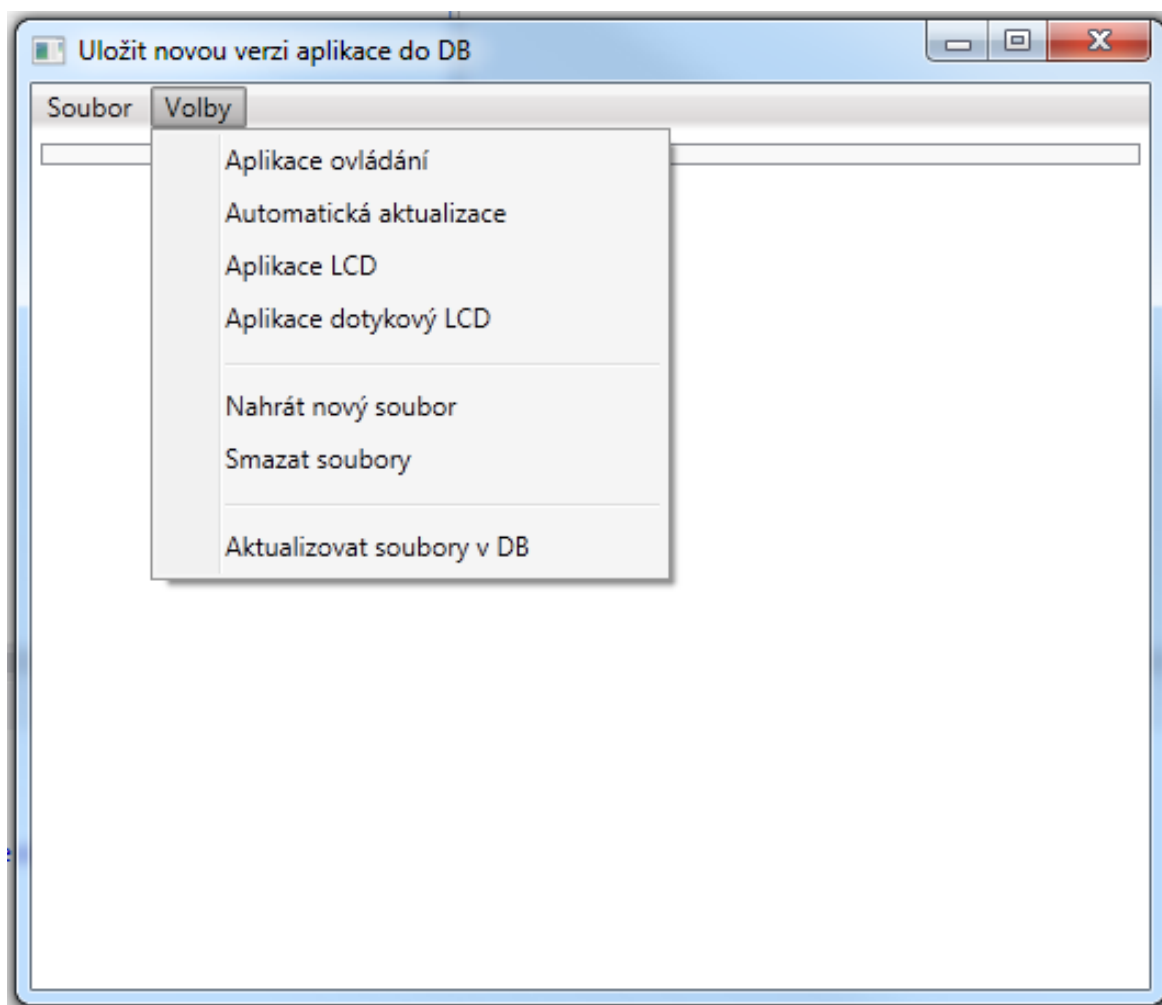
Ve službě *FileTransferService.svc*, uvedené v příloze bych rád upozornil na to, jakým způsobem se pomocí cyklu `while` přenáší stream dat z klienta na server. Je zde vidět, že je stream čtený dokud protistrana odesílá data.

10 SPRÁVA VERZÍ APLIKACÍ

Jak jsem již zmínil, klientskou aplikaci *ControlInterface* bude využívat několik uživatelů, stejně tak, jako aplikace pro zobrazení *LcdInterface* a *TouchLcdInterface* má být spuštěna z několika PC. Ze zkušenosti vím, že je vždy dobré počítat s možností oprav nebo zdokonalení aplikací pro uživatele a bývá náročné aktualizovat aplikace uživatelům pomocí emailu nebo ručně. Z toho důvodu jsem vyvinul systém na ukládání nových verzí a jejich automatickou aktualizaci při spuštění nebo za běhu dané klientské aplikace.

V síťovém prostředí ABB, které je svázané poměrně přísnými pravidly jsem zvolil aktualizaci pomocí pomocných aplikací, komunikujících s databází prostřednictvím webových služeb.

10.1 Uložení



Obr. 19. Ukládání nových verzí klientské aplikace

Na obrázku obr. 19 je náhled na aplikaci *AppSaver* a její nabídku. Podotýkám, že tato

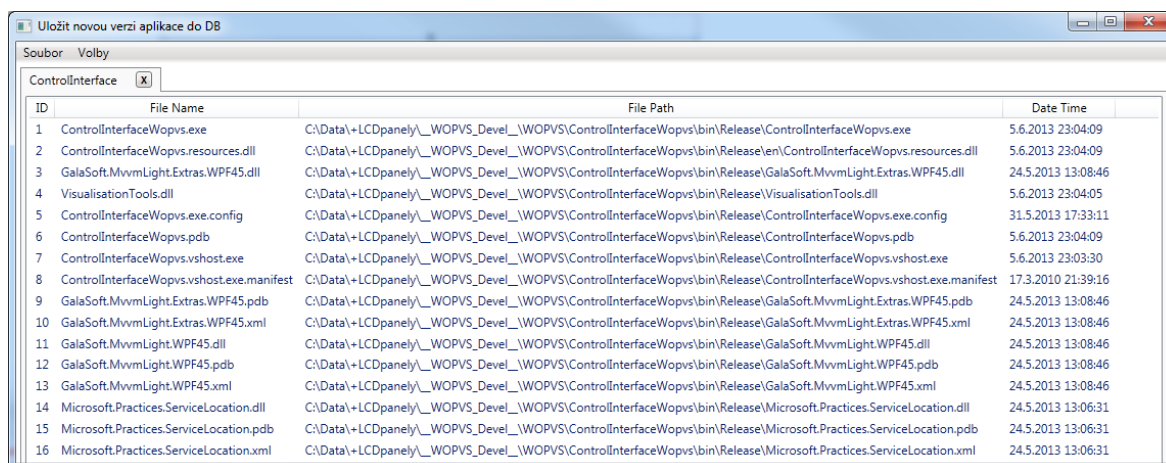
AppSaver slouží výhradně programátorovi, nikoli klientům a jejím účelem je uložit aktuální soubory klientských aplikací po případných úpravách.

Z rozbalené nabídky *Volby* je patrné, že *AppSaver* slouží k ukládání

- Aplikace ovládání
- Automatické aktualizace
- Aplikace LCD
- Aplikace dotykového LCD

Programátor po volbě aplikace z nabídky, například Aplikaci ovládání *ControlInterface*. Vypis souborů *ControlInterface* je na obrázku obr. 20. Pomocí položek z nabídky na obrázku obr. 19 *Nahrát nový soubor* a *Smazat soubory* lze měnit obsah seznamu a tedy i soubory, které se klientům v případě potřeby aktualizují. Položkou nabídky *Aktualizovat soubory v DB* může programátor aktualizovat soubory v DB na všech otevřených kartách zároveň.

Na obrázku obr. 20 je ve sloupci *DateTime* uložený datum a čas vytvoření souboru, který má význam pro aktualizaci, popsané v následující kapitole.



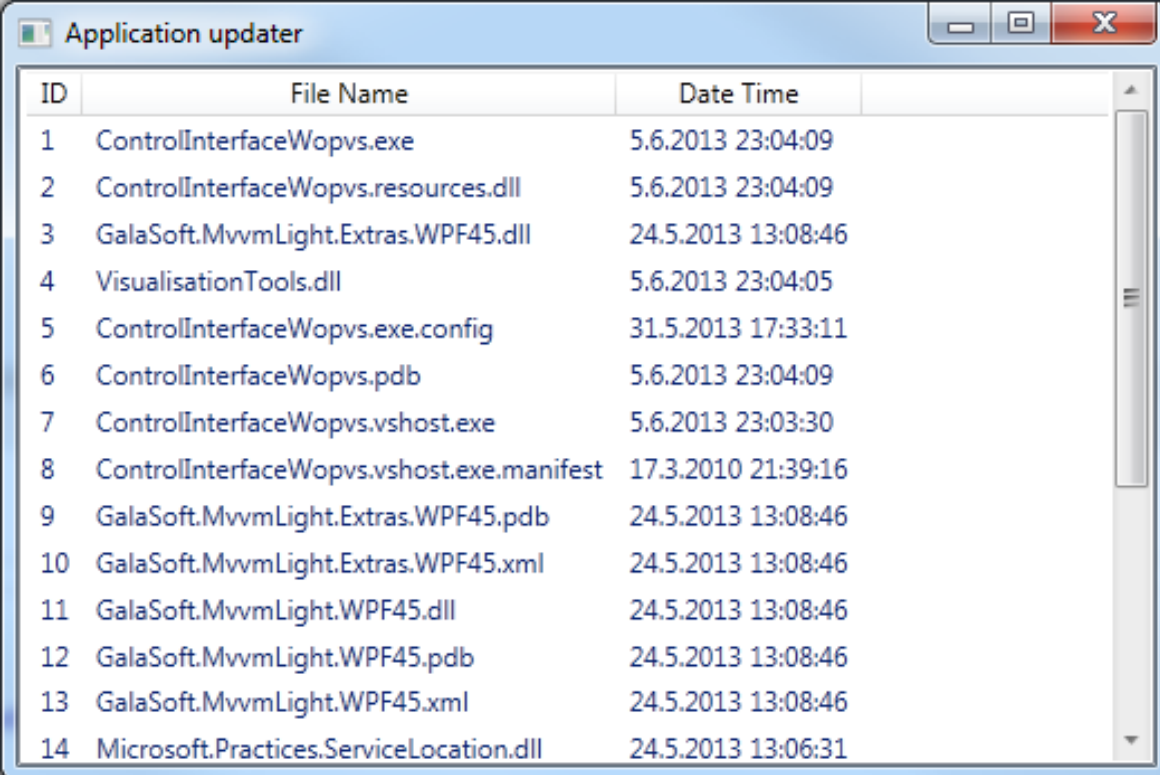
ID	File Name	File Path	Date Time
1	ControlInterfaceWopvs.exe	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\ControlInterfaceWopvs.exe	5.6.2013 23:04:09
2	ControlInterfaceWopvs.resources.dll	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\en\ControlInterfaceWopvs.resources.dll	5.6.2013 23:04:09
3	GalaSoft.MvvmLight.Extras.WPF45.dll	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.Extras.WPF45.dll	24.5.2013 13:08:46
4	VisualisationTools.dll	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\VisualisationTools.dll	5.6.2013 23:04:05
5	ControlInterfaceWopvs.exe.config	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\ControlInterfaceWopvs.exe.config	31.5.2013 17:33:11
6	ControlInterfaceWopvs.pdb	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\ControlInterfaceWopvs.pdb	5.6.2013 23:04:09
7	ControlInterfaceWopvs.vshost.exe	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\ControlInterfaceWopvs.vshost.exe	5.6.2013 23:03:30
8	ControlInterfaceWopvs.vshost.exe.manifest	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\ControlInterfaceWopvs.vshost.exe.manifest	17.3.2010 21:39:16
9	GalaSoft.MvvmLight.Extras.WPF45.pdb	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.Extras.WPF45.pdb	24.5.2013 13:08:46
10	GalaSoft.MvvmLight.Extras.WPF45.xml	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.Extras.WPF45.xml	24.5.2013 13:08:46
11	GalaSoft.MvvmLight.WPF45.dll	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.WPF45.dll	24.5.2013 13:08:46
12	GalaSoft.MvvmLight.WPF45.pdb	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.WPF45.pdb	24.5.2013 13:08:46
13	GalaSoft.MvvmLight.WPF45.xml	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\GalaSoft.MvvmLight.WPF45.xml	24.5.2013 13:08:46
14	Microsoft.Practices.ServiceLocation.dll	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\Microsoft.Practices.ServiceLocation.dll	24.5.2013 13:06:31
15	Microsoft.Practices.ServiceLocation.pdb	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\Microsoft.Practices.ServiceLocation.pdb	24.5.2013 13:06:31
16	Microsoft.Practices.ServiceLocation.xml	C:\Data\+LCDpanel__WOPVS_Devel__\WOPVS\ControlInterfaceWopvs\bin\Release\Microsoft.Practices.ServiceLocation.xml	24.5.2013 13:06:31

Obr. 20. Ukládání nových verzí klientské aplikace

10.2 Aktualizace

Při spuštění *Aplikace ovládání* a *Automatické aktualizace*, nebo v případě *Aplikace LCD* a *Aplikace dotykového LCD* jednou za stanovenou periodu, pomocný program *Application updater*, uvedený na obrázku obr. 21, kontroluje podle datumu vytvoření souboru v aktuálním adresáři vůči datumu vytvoření souboru v databázi. Jestliže má soubor na disku starší datum vytvoření, nežli je datum daného souboru uvedené v

databázi, nebo v databázi existuje soubor, který v aktuálním adresáři není, proběhne aktualizace. Aktualizují se pouze soubory, které aktualizaci potřebují.



The screenshot shows a window titled "Application updater" with a table of files. The table has three columns: "ID", "File Name", and "Date Time". The files listed are:

ID	File Name	Date Time
1	ControlInterfaceWopvs.exe	5.6.2013 23:04:09
2	ControlInterfaceWopvs.resources.dll	5.6.2013 23:04:09
3	GalaSoft.MvvmLight.Extras.WPF45.dll	24.5.2013 13:08:46
4	VisualisationTools.dll	5.6.2013 23:04:05
5	ControlInterfaceWopvs.exe.config	31.5.2013 17:33:11
6	ControlInterfaceWopvs.pdb	5.6.2013 23:04:09
7	ControlInterfaceWopvs.vshost.exe	5.6.2013 23:03:30
8	ControlInterfaceWopvs.vshost.exe.manifest	17.3.2010 21:39:16
9	GalaSoft.MvvmLight.Extras.WPF45.pdb	24.5.2013 13:08:46
10	GalaSoft.MvvmLight.Extras.WPF45.xml	24.5.2013 13:08:46
11	GalaSoft.MvvmLight.WPF45.dll	24.5.2013 13:08:46
12	GalaSoft.MvvmLight.WPF45.pdb	24.5.2013 13:08:46
13	GalaSoft.MvvmLight.WPF45.xml	24.5.2013 13:08:46
14	Microsoft.Practices.ServiceLocation.dll	24.5.2013 13:06:31

Obr. 21. Čtení nových verzí klientské aplikace

11 APLIKACE OVLÁDÁNÍ

Aplikace ovládání je klíčovým prvkem celého systému. S pomocí této aplikace mohou administrátoři LCD panelů, dotykových i obyčejných, nastavovat co se má v jakém okamžiku zobrazit, případně jaká je struktura nabídky dotykového LCD.

S pomocí návrhového vzoru MVVM, je rozhraní psané pomocí jazyka XAML jednoduché a intuitivní. V této kapitole nejprve uvedu na příkladu použití návrhového vzoru MVVM a poté se budu věnovat popisu samotné aplikace a jejích funkcí.

11.1 MVVM v aplikaci ovládání

11.1.1 RelayCommand

Každý pohled aplikace má prázdný přidružený soubor s ovládacím zdrojovým kódem (ve WPF aplikacích se standardně používá tento soubor k obsluze událostí a příkazů), až na několik řádků automaticky generovaného kódu vývojovým prostředím. Ve skutečnosti by bylo možné tyto soubory, v případě použití návrhového vzoru MVVM, úplně odstranit a aplikace by se korektně kompilovala.

Obsluha událostí je zajištěna pomocí objektů *ICommand* zveřejněných v odpovídajících třídách *ViewModel* . Ve formulářích *View* jsou objekty *ICommand* volané prostřednictvím vazeb na vlastnosti *ViewModel* typu *ICommand* .

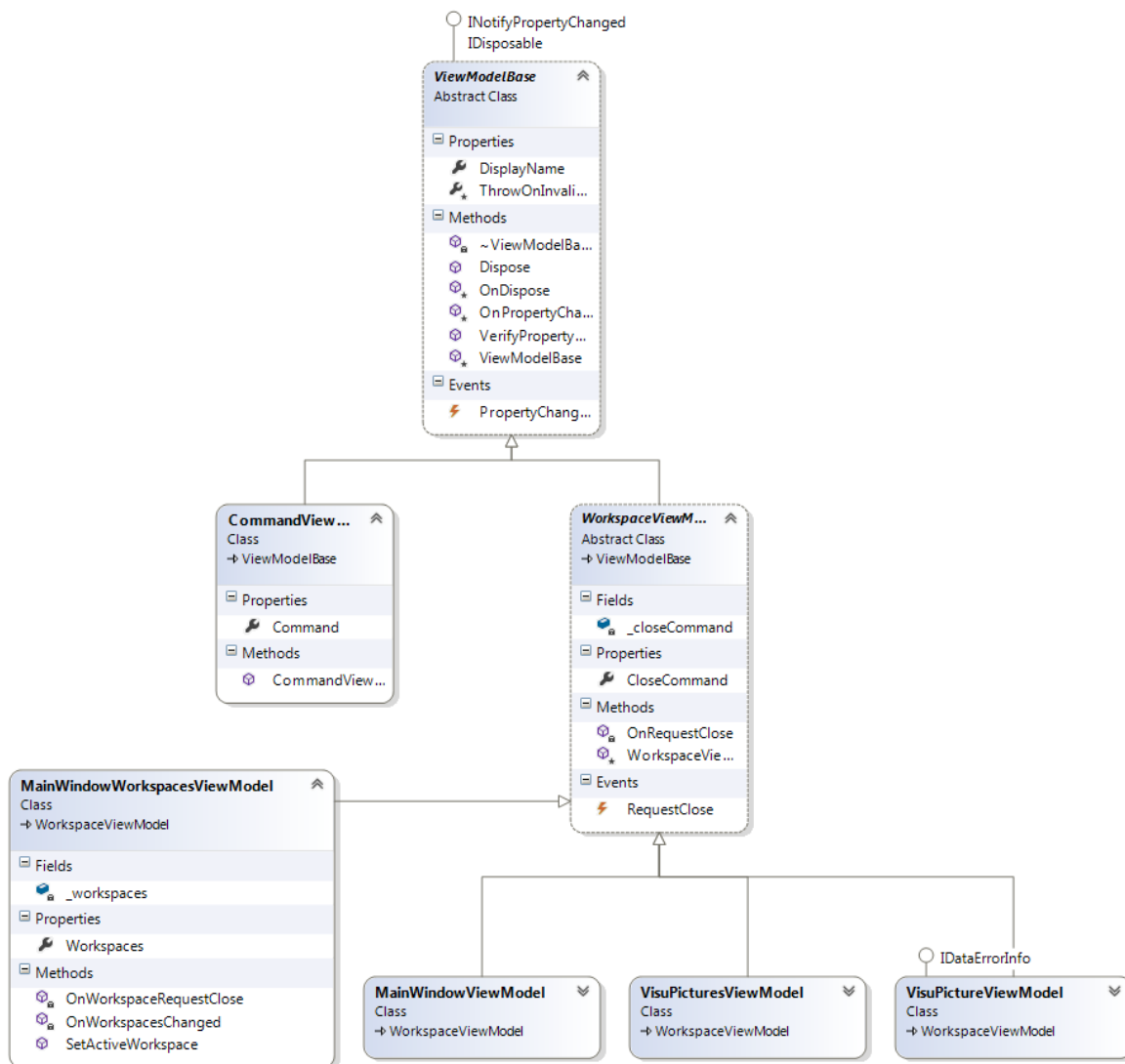
V okamžiku, kdy *ViewModel* vytvoří instanci vlastnosti typu *ICommand* , nově vytvořený objekt typicky využívá objekt *ViewModel* k vykonání příslušné akce. Jedna možnost implementace vzoru je vytvoření soukromé zapouzdřené třídy uvnitř třídy *ViewModel* , takže příkazy mají přístup k soukromým členům obsaženým v tomto *ViewModel* a "neznečišťují" jmenný prostor. Tato vnořená třída implementuje rozhraní *ICommand* a odkazuje na objekty obsažené ve *ViewModel* prostřednictvím konstrukturu. Tímto způsobem, kdy každá třída *ViewModel* obsahuje vnořenou třídu, začne množství kódu narůstat a zvyšuje se riziko nečekaných chyb.

V příloze uvedená třída *RelayCommand.cs* tento problém velice elegantně řeší. *RelayCommand* umožňuje vnoření příkazové logiky prostřednictvím delegátů předaných do konstrukturu. Tento přístup umožní vložit veškerou příkazovou obslužnou logiku do tříd *ViewModel* , bez zbytečného kódu navíc. *RelayCommand* je ve skutečnosti zjednodušená variace *DelegateCommand* , standardně obsažené v *Microsoft Composite Application Library* .

11.1.2 Hierarchie ViewModel

Většina tříd *ViewModel* využívá stejnou základní funkcionalitu. Obvykle potřebují implementaci rozhraní *INotifyPropertyChanged* , uživatelsky přívětivé rozhraní pro zobrazení ve *View* , například název pro zobrazení a standardní příkazy, například schopnost

se zavřít. Tyto problémy se přirozeně rozdělují do základní třídy *ViewModelBase* a nově tvořené, účelové, třídy *ViewModel*. Z důvodu přehlednosti, udržitelnosti a snadné rozšiřitelnosti aplikací je velmi vhodné pro tyto společné úlohy využít dědičnosti tříd, jak je uvedeno například na obrázku obr. 22.



Obr. 22. Hierarchie tříd ViewModel

Pro návrhový vzor MVVM je vytvoření základní třídy *ViewModelBase* ovšem nutností. Samozřejmě je možné zvolit i jinou cestu, například prostřednictvím menších tříd a odkazů na ně, ale právě MVVM doporučuje tuto cestu.

11.1.3 Třída *ViewModelBase*

Tato třída je kořenovou třídou hierarchie (výpis je uvedený v příloze). Implementuje rozhraní *INotifyPropertyChanged* a vlastnost *DisplayName*. Rozhraní *INotifyPropertyChanged* obsahuje událost *PropertyChanged*. Kdykoli má vlastnost na libovolném

objektu *ViewModel* novou hodnotu, může být vyvolána událost *PropertyChanged*, aby upozorňovala vazby systému WPF na novou hodnotu. Po obdržení upozornění, je vlastnost s novou hodnotou zařazena jako událost k aktualizaci uživatelského rozhraní.

Aby WPF věděl, která vlastnost ve *ViewModel* je změněná, zprostředkovává třída *PropertyChangedEventArgs* vlastnost *PropertyName* typu *String*. Zde je třeba být obezřetný při předávání názvu typů, aby byl název vlastnosti dodržen.

11.1.4 Třída *CommandViewModel*

Tato třída, uvedená v příloze, poskytuje rozhraní *Command* typu *ICommand*. Třída *MainWindowViewModel* převádí kolekci těchto objektů na vlastnosti.

11.1.5 Třída *MainWindowViewModel*

Třída *MainWindowViewModel* (uvedená v příloze) zajišťuje zprávu vytvořených potomků *ViewModel*, pro něž je kontejnerem. Umožňuje vytvoření a zrušení nového objektu.

11.1.6 Třída *MainWindowWorkspacesViewModel*

Tato třída, uvedená v příloze, má na starost předání celé kolekce otevřených objektů *ViewModel*. Při tvorbě nových objektů je předávána v konstruktoru a jejím účelem je umožnit otevření dalších potomků hlavního okna z již otevřených potomků. Tyto nové objekty tak mohou být správně zařazené do celé kolekce.

11.2 Organizace projektu

Aplikace ovládání již obsahuje poměrně velké množství souborů tříd, zdrojů, uživatelských ovládacích prvků a podobně. Z toho důvodu, a také v souladu s doporučením návrhového vzoru MVVM, jsem soubory řadil do následujících složek:

Service References obsahuje generované soubory s odkazy na webovou službu WCF

ClassDiagrams obsahuje graficky zpracované třídy a vazby mezi nimi

DataAccess obsahuje třídy pro komunikaci s databází, zejména se jedná o repozitáře objektů obsahující kolekce daných typů; dále obsahuje argumenty události "vlození nového objektu do kolekce"

LocalResources obsahuje soubory s jazykovými mutacemi, vždy ke každému *ViewModel* resp. *View*

Models obsahuje konkrétní objekty pro práci s aplikací ovládání a objekty pro zobrazení; tyto objekty dědí rozhraní *IDataErrorInfo* a překrývají část implementace,

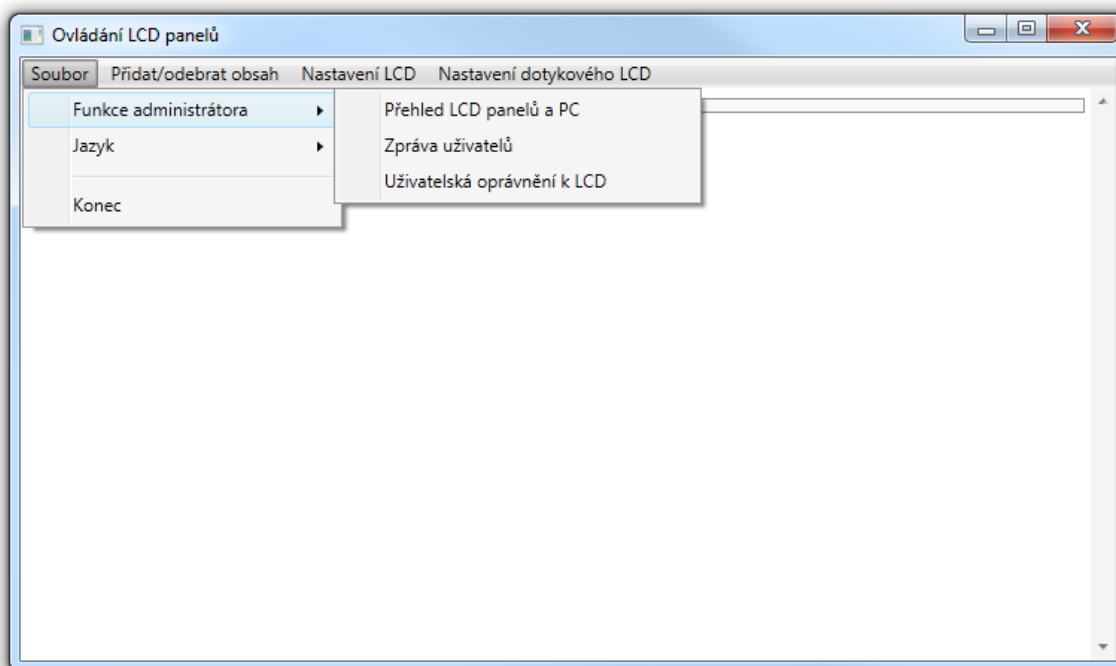
čímž jsou objekty schopné zkontrolovat stav naplnění svých vlastností. V příloze je uvedený příklad základní třídy *VisuBaseObject.cs*, pro některé vizualizační prvky

Resources obsahuje pomocné soubory, například obrázky nebo ikony, které využívá aplikace ovládání při kompilaci

ViewModels obsahuje třídy *ViewModels* zajišťující veškerou aplikační logiku komunikace objektů *Models* s rozhraním pro uživatele *Views*

11.3 Popis nabídek

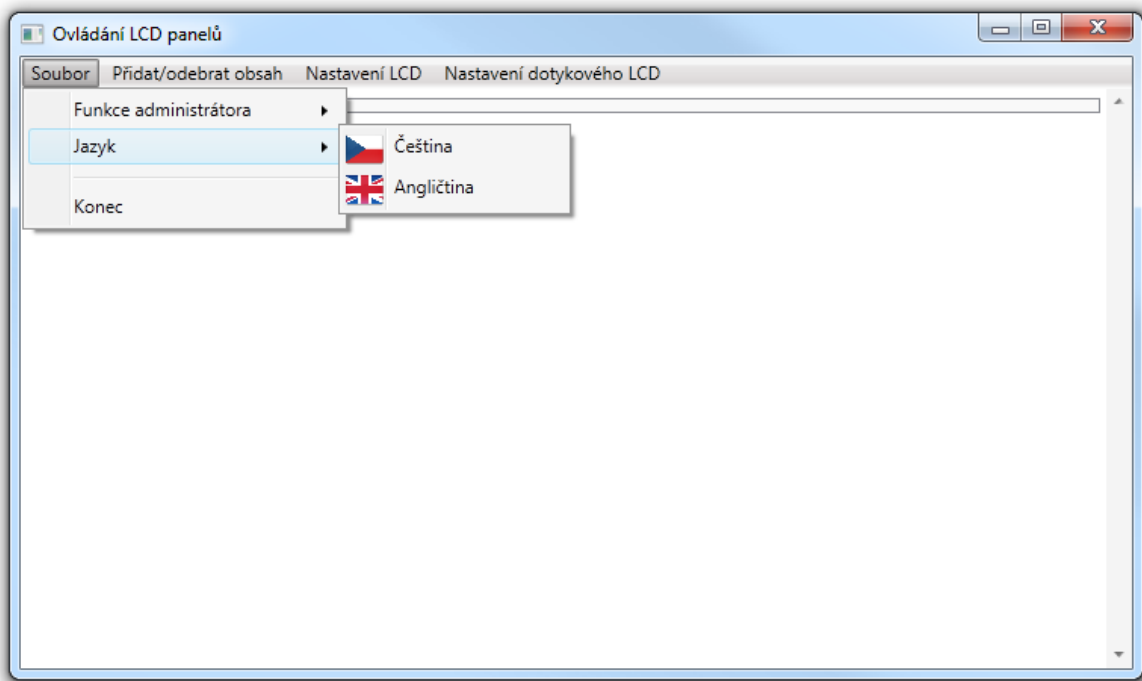
Na obrázku obr. 23 je struktura nabídky *Soubor*. Podnabídka *Funkce administrátora* je přístupné pouze pro administrátora se zvláštním oprávněním, neboť obsahuje možnost tvorby LCD panelů a PC, zprávu uživatelů a jejich oprávnění. Jedná se o akce, které vyžadují větší znalosti a odpovědnost za provedené úpravy.



Obr. 23. Nabídka Soubor / Funkce administrátora

Obrázek obr. 24 obsahuje veřejně přístupnou nabídku s volbou jazyka prostředí. V této, první verzi, jsem vytvořil českou a anglickou jazykovou mutaci. Ovšem vzhledem k použitým principům popsaným v kapitole Lokalizace aplikace WPF, je doplnění dalších jazyků velmi snadnou záležitostí.

Nabídka *Přidat/odebrat obsah* na obrázku obr. 25 obsahuje nejprve možnost otevření přehledů:



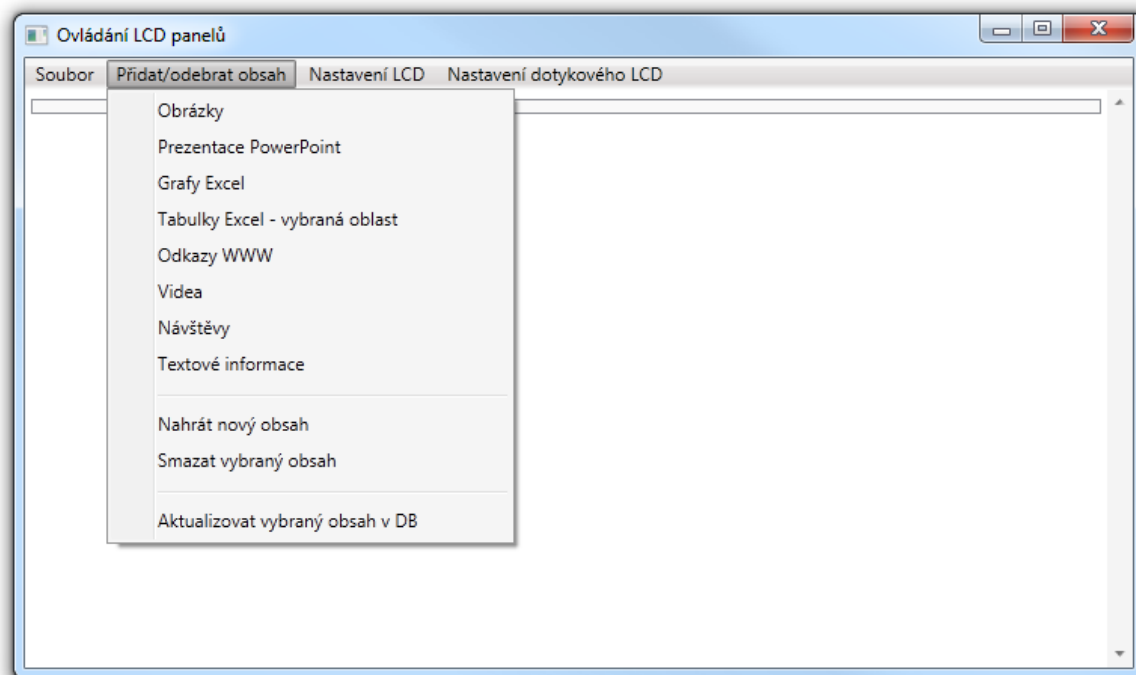
Obr. 24. Nabídka Soubor / Jazyk

- Obrázky
- Prezentace Power point
- Grafy Excel
- Tabulky Excel - vybraná oblast
- Odkazy WWW
- Videá
- Návštěvy
- Textové informace

Tyto přehledy zahrnují vždy seznam daných prvků (dle názvu), které lze zařadit do cyklů nebo do nabídek pro zobrazení na LCD panelech.

Položky nabídky pracují s aktuálně otevřeným přehledem. Tyto funkce mohly být zařazeny do jednotlivých přehledů například v podobě tlačítek, ale vzhledem k možnému velkému rozsahu seznamů v přehledech jsem zvolil tuto možnost z důvodu zpřehlednění a větší jednotnosti ovládání.

Nahrát nový obsah vkládá do otevřeného přehledu na popředí nový prvek



Obr. 25. Nabídka Přidat/odebrat obsah

Smazat vybraný obsah odebere z otevřeného přehledu na popředí vybrané prvky

Aktualizovat vybraný obsah aktualizuje dostupné soubory z lokálních nebo síťových disků dle uložené cesty daných prvků, v přehledu na popředí

Položky nabídky *Nastavení LCD*, zobrazené na obrázku obr. 26, zahrnují:

Přehled a rozvržení je stejný formulář jako v nabídce *Funkce administrátora* s tím, že některé funkce jsou omezené. Obsahuje možnost nastavení layoutu LCD panelů.

Cykly LCD umožňuje nastavení cyklů zobrazení v jednotlivých oblastech panelů.

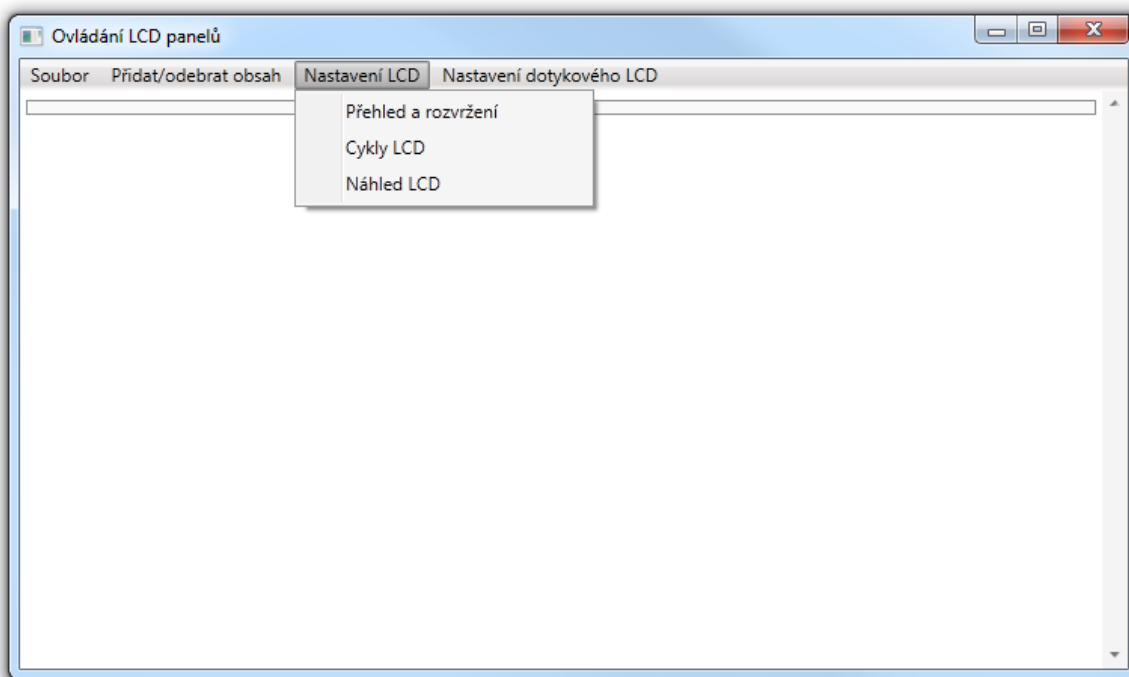
Náhled LCD možnost prohlídky vytvořených cyklů v aplikaci ovládání

Na obrázku obr. 27 je nabídka určená k ovládání dotykových LCD panelů.

Nabídka dotykového LCD tvorba základní struktury nabídky na LCD

Položky nabídky dotykového LCD umožňuje zařazení dokumentů do nabídek

Náhled dotykového LCD slouží k zobrazení aktuálního nastavení uvnitř aplikace ovládání



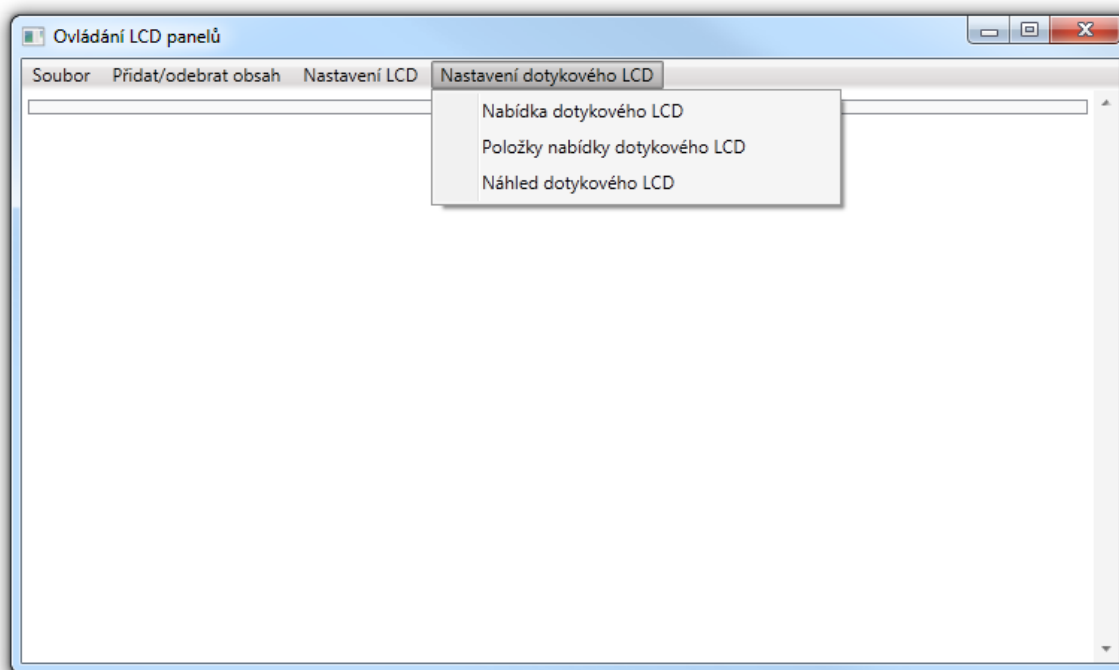
Obr. 26. Nabídka Nastavení LCD

V následujících podkapitolách uvedu několik příkladů funkcí aplikace ovládání. Výčet nebude kompletní z důvodu velkého rozsahu a možností celého systému zobrazení informací na LCD panelech. Dále je zřejmé, že bude třeba možnosti aplikace dále rozšiřovat na základě požadavků uživatelů, nebo případně upravovat v závislosti na nových technologiích a vývoji bezpečnostní a softwarové politiky ve firmě.

11.3.1 Přehled objektů pro zobrazení - obrázky

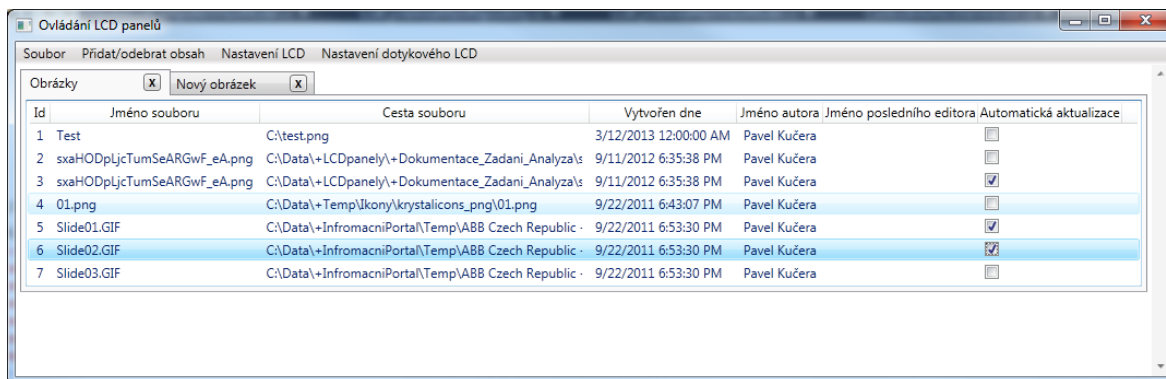
Na obrázku obr. 28 je několik vložených souborů do databáze, připravených pro zařazení do cyklu a zobrazení na LCD panelech.

Kromě jména souboru zde nechávám zobrazené také *ID* souboru generované databází. Není to zbytečné, neboť zkušenost ukázala, že uživatel při komunikaci nebo editaci mnohem rychleji a jednoznačněji pracuje s *ID* obrázku. Dále je zde vidět *Cesta souboru*. Podle té uživatel pozná, zda v dané chvíli může provést aktualizaci objektu do databáze a kde se soubor nachází. Ve sloupci *Vytvořen dne* je datum a čas vytvoření souboru. Ten je velmi důležitý zejména při zobrazování na LCD panelech, neboť v případě velkých videí nebo prezentací powerpoint, zobrazovací program na základě porovnání datumu vytvoření souboru v databázi a v dočasném adresáři na disku rozhoduje, zda bude soubor znovu číst z databáze či nikoli. Posledním sloupcem, na který bych rád upozornil, je sloupec *Automatická aktualizace*. Ten slouží aplikaci *DataUpdateMachine* (automatická aktualizace dat) jako informace o tom, které objekty je třeba aktualizovat



Obr. 27. Nabídka Nastavení dotykového LCD

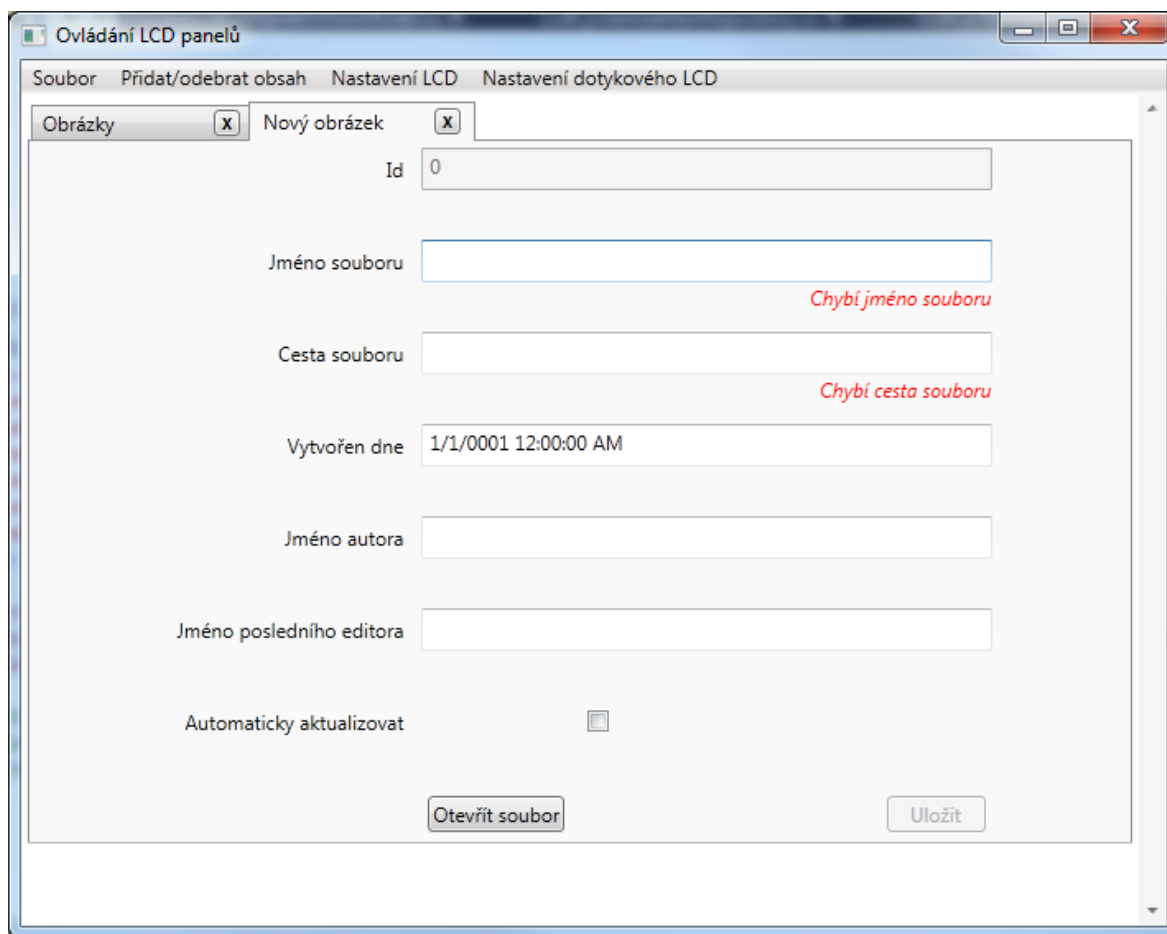
v pravidelných intervalech, nastavených jako naplánované úlohy operačního systému.



Obr. 28. Seznam obrázků

Formulář pro vložení nového obrázku je na obrázku obr. 29. Zde bych rád upozornil na červenou kurzívou psaný text pod textovými poli *Jméno souboru* a *Cesta souboru*. Jedná se o výsledek vyhodnocení validity objektu obrázku pomocí rozhraní *IDataErrorInfo*. Kód je k nahlédnutí uvedený v příloze *VisuBaseObject.cs*; objekt obrázek (*VisuPicture*) totiž dědí tyto vlastnosti od *VisuBaseObject*.

Uživatel do tohoto formuláře může zapsat jméno souboru ručně, mnohem jednodušší variantu doplnění souboru do databáze mu ovšem poskytuje tlačítko *Otevřít soubor*, které všechna potřebná pole automaticky vyplní na základě vybraného souboru na lokálním nebo síťovém úložišti.



The screenshot shows a software window titled "Ovládání LCD panelů" with a menu bar containing "Soubor", "Přidat/odebrat obsah", "Nastavení LCD", and "Nastavení dotykového LCD". Below the menu bar are two tabs: "Obrázky" and "Nový obrázek". The "Nový obrázek" tab is active and contains a form with the following fields and controls:

- "Id": A text input field containing the value "0".
- "Jméno souboru": A text input field with a red error message "Chybí jméno souboru" below it.
- "Cesta souboru": A text input field with a red error message "Chybí cesta souboru" below it.
- "Vytvořen dne": A text input field containing the date and time "1/1/0001 12:00:00 AM".
- "Jméno autora": A text input field.
- "Jméno posledního editora": A text input field.
- "Automaticky aktualizovat": A checkbox that is currently unchecked.
- At the bottom of the form are two buttons: "Otevřít soubor" and "Uložit".

Obr. 29. Formulář pro vložení nového obrázku do databáze

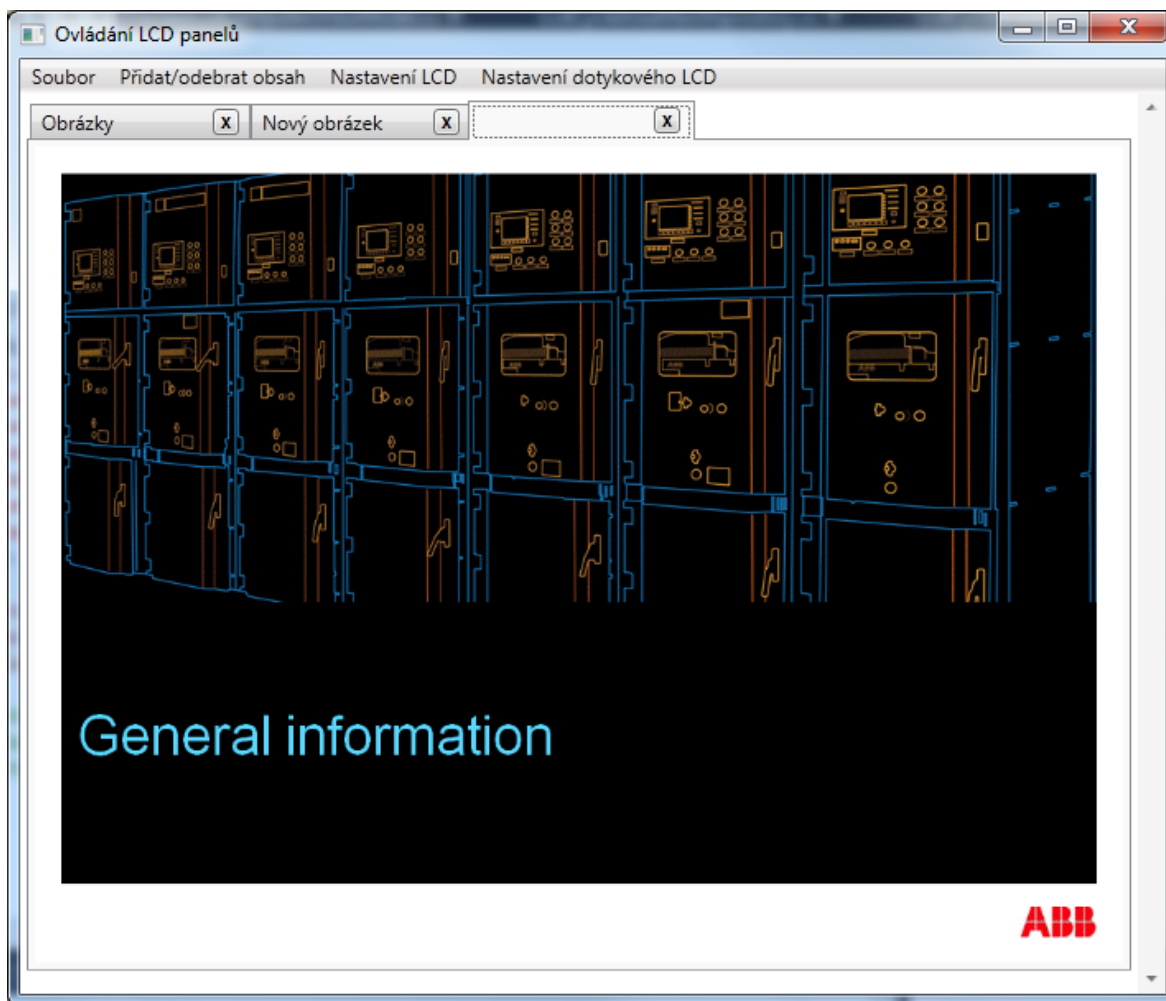
Rád bych také upozornil na vlastnost seznam objektů otevřít objekt k náhledu. Pomocí dvojkliku myši na položku seznamu se otevře náhled objektu tak, jak je uložený v databázi (obr. 30).

11.3.2 Rozvržení LCD panelů

Přehled rozvržení LCD panelů je dostupný z nabídky *Soubor/Funkce administrátora* a z nabídky *Nastavení LCD*, přičemž běžný administrátor LCD panelů resp. uživatel nemá k dispozici možnost přidávat nová rozvržení, může pouze upravit existující. Tlačítko *Přidat rozvržení LCD* (obr. 31) tudíž nemá přístupné.

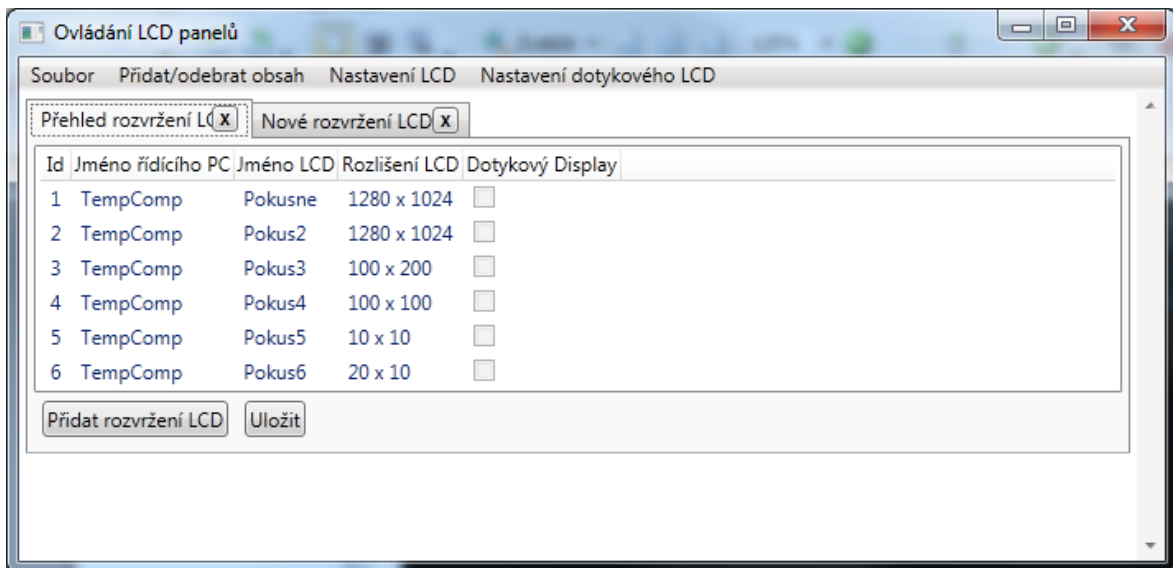
Z přehledu lze získat informace o názvu LCD panelu, ke kterému PC je připojený, jaké má rozlišení a zda se jedná o dotykový LCD panel či nikoli.

Rozsáhlejší formulář pro návrh nového rozvržení LCD panelu je vidět na obrázku obr. 32. Stejně jako v případě vytváření vizualizačních objektů, je i tento formulář kontrolován pomocí *IDataErrorInfo* a pouze v případě validity zadaných údajů, lze nové rozvržení uložit. V tomto formuláři jsou vazby specifické v tom, že více než jeden zadaný údaj tvoří chybový výsledek. Například v případě zatržení check boxu *Display*



Obr. 30. Příklad zobrazení obrázku v aplikaci ovládání

má doplňkovou zobrazovací oblast, musí být zároveň nenulové hodnoty šířky a výšky doplňkové oblasti. Aby bylo dosaženo správné funkcionality, je třeba ve vlastnostech `LcdLayoutViewModel` doplnit události `OnPropertyChanged` křížem do všech souvisejících vlastností (viz. následující krátký příklad):



Obr. 31. Přehled vytvořených rozvržení LCD

```

public bool HasAdditionalArea
{
    get { return _lcdLayout.HasAdditionalArea; }
    set
    {
        if (value == _lcdLayout.HasAdditionalArea)
            return;
        _lcdLayout.HasAdditionalArea = value;
        base.OnPropertyChanged("HasAdditionalArea");
        base.OnPropertyChanged("AdditionalAreaResolutionH");
        base.OnPropertyChanged("AdditionalAreaResolutionV");
    }
}
public int AdditionalAreaResolutionH
{
    get { return _lcdLayout.AdditionalAreaResolutionH; }
    set
    {
        if (value == _lcdLayout.AdditionalAreaResolutionH)
            return;
        _lcdLayout.AdditionalAreaResolutionH = value;
        base.OnPropertyChanged("AdditionalAreaResolutionH");
        base.OnPropertyChanged("HasAdditionalArea");
    }
}

```

Ve formuláři návrh nového rozvržení lze pomocí tlačítka *Nový*, napravo od combo boxu *Jméno ovládacího PC*, doplnit nové PC, ke kterému je LCD panel fyzicky připojený. Po stisku tlačítka se otevře velmi jednoduchý formulář (obr. 33), kam lze jméno počítače zapsat. Toto jméno hraje při zobrazení na LCD panelech velmi zásadní roly, neboť se jedná o síťové jméno počítače a na základě automatické identifikace z kódu lze potom na daném PC otevřít všechna rozvržení LCD panelů.

11.3.3 Cykly LCD panelu

Jak bylo již zmíněno, na každém LCD panelu musí být zobrazená hlavní zobrazovací oblast a může být zobrazena doplňková zobrazovací oblast a běžící text. Každý z těchto kontejnerů, dělících celkovou plochu LCD panelu na oblasti definované administrátorem, může zobrazovat svůj vlastní cyklus prvků. Každý cyklus je vlastně sekvence vizuálních objektů, které se zobrazují po definovaný časový úsek (v případě například obrázků), nebo dokud neskončí (například v případě videí).

Na obrázku obr. 34 je přehled LCD panelů a jejich rozvržení. Tento formulář umožňuje editovat cykly jednotlivých oblastí, tedy vkládat do cyklů vizuální prvky, které jsou uloženy v databázi. Výběrem řádku se mění dostupnost tlačítek charakterizujících oblasti v pravé části formuláře.

Formuláře, obsahující přehled definovaných cyklů (viz. obr. 35) jsou pro hlavní a doplňkovou oblast i pro běžící text velmi podobné. Umožňují pomocí tlačítek pod seznamem prvků cyklu

Přidat položku - otevře formulář pro vložení nového vizuálního prvku

Odstranit položky - odstraní všechny vybrané položky

Nahoru - posune vybranou položku nahoru (k začátku cyklu)

Dolu - posune vybranou položku dolů (ke konci cyklu)

Uložit - uloží cyklus

V prvním sloupci seznamu je *Pořadí* ve kterém se od začátku budou jednotlivé položky zobrazovat na LCD panelu. Druhý sloupec obsahuje informaci o vizuální objektu, tedy o jaký druh objektu se jedná a jeho krátký název nebo popis. Zda má položka přednost v cyklu před jiným, je limitovaná datumem a časem a její délka zobrazení je v následujících sloupcích.

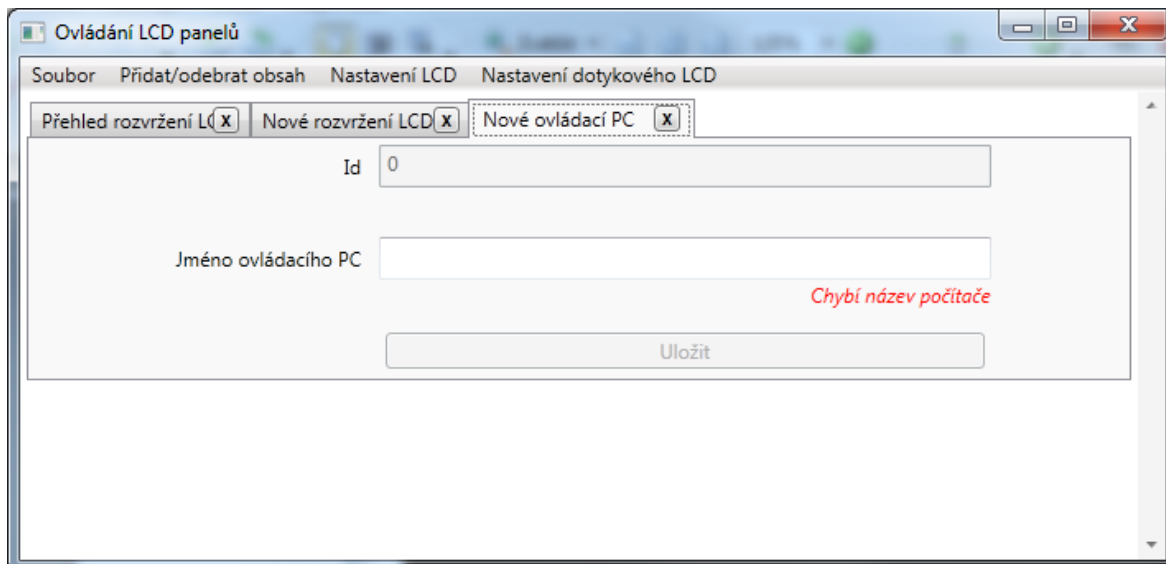
Ve formuláři *Nový prvek* (obrázek obr. 36) lze doplnit další prvek do cyklu. Může se jednat o libovolný prvek (obrázek, prezentace, graf, ...), uložený v databázi. Jestliže uživatel potřebuje přidat prvek, který dosud v databázi není, může z tohoto formuláře, vždy tlačítkem *Nový* příslušné položky, uložit prvek do databáze a přidat do cyklu. Pod tlačítky *Nový* se skrývá stejný standardní formulář pro přidávání vizuálních objektů jako v nabídce *Přidat/odebrat obsah*.

The screenshot shows a software window titled "Ovládání LCD panelů" with a menu bar containing "Soubor", "Přidat/odebrat obsah", "Nastavení LCD", and "Nastavení dotykového LCD". Below the menu bar are two tabs: "Přehled rozvržení LCD (X)" and "Nové rozvržení LCD (X)". The main area contains the following configuration fields:

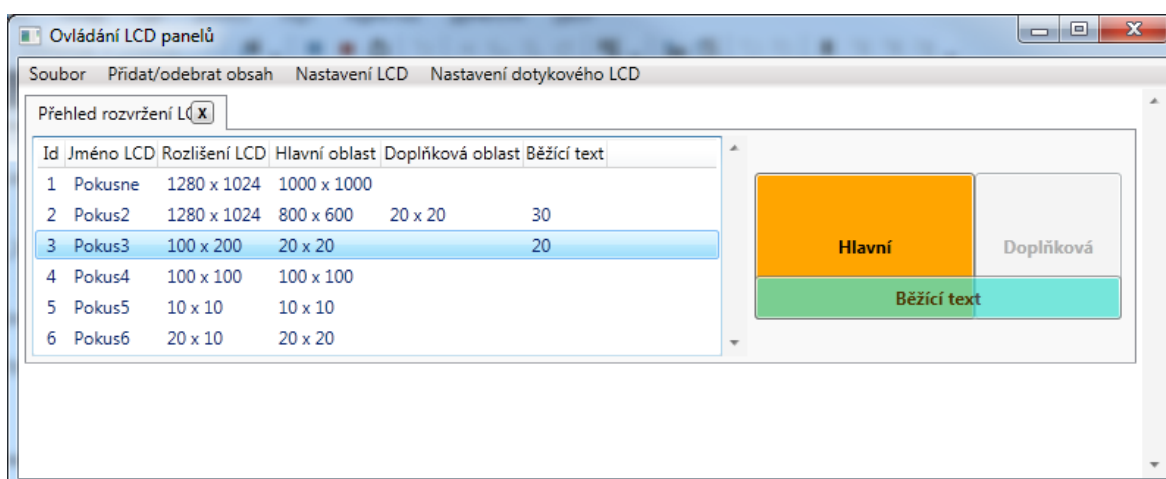
- Id:** Text input field containing "0".
- Jméno ovládacího PC:** A dropdown menu with a "Nový" button next to it. Below the dropdown is the red error message: "Není nastavený ovládací počítač LCD panelu".
- Jméno LCD panelu:** Text input field. Below it is the red error message: "Není zadáno jméno LCD panelu".
- Horizontální rozlišení:** Text input field containing "0". Below it is the red error message: "Není zadáný horizontální rozměr".
- Vertikální rozlišení:** Text input field containing "0". Below it is the red error message: "Není zadáný vertikální rozměr".
- Horizontální pozice:** Text input field containing "0".
- Vertikální pozice:** Text input field containing "0".
- Dotykový displej:** A checkbox that is currently unchecked.
- Šířka hlavní zobrazovací oblasti:** Text input field containing "0". Below it is the red error message: "Není zadaná šířka hlavní oblasti".
- Výška hlavní zobrazovací oblasti:** Text input field containing "0". Below it is the red error message: "Není zadaná výška hlavní oblasti".
- Displej má doplňkovou zobrazovací oblast:** A checkbox that is currently unchecked.
- Šířka doplňkové oblasti:** Text input field containing "0".
- Výška doplňkové oblasti:** Text input field containing "0".
- Displej má běžící text:** A checkbox that is currently unchecked.
- Výška běžícího textu:** Text input field containing "0".

An "Uložit" (Save) button is located at the bottom right of the window.

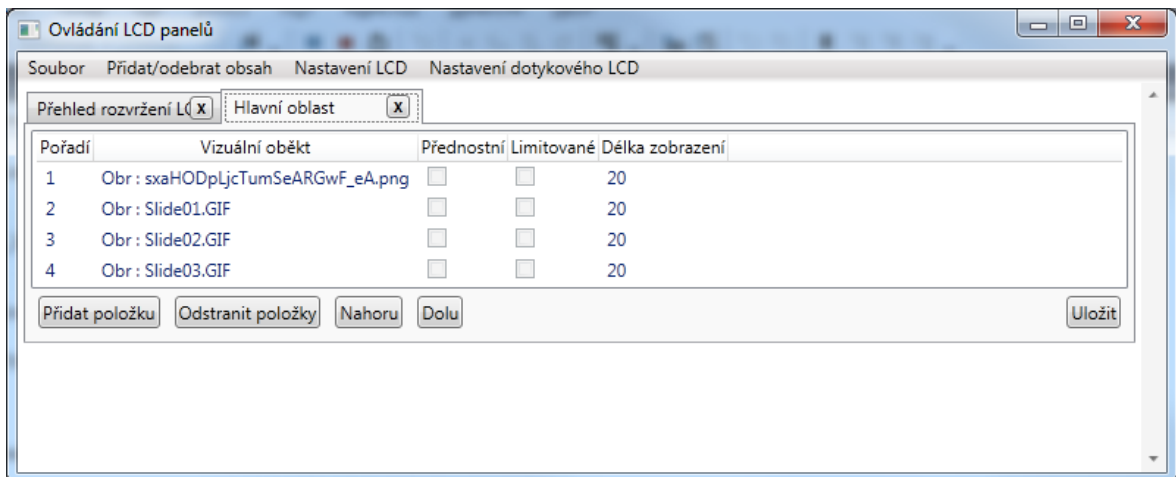
Obr. 32. Návrh nového rozvržení LCD



Obr. 33. Doplnění nového ovládacího PC k rozvržení



Obr. 34. Přehled rozvržení LCD panelů



Obr. 35. Hlavní cykly LCD panelu

Ovládání LCD panelů

Soubor Přidat/odebrat obsah Nastavení LCD Nastavení dotykového LCD

Přehled rozvržení L(x) Hlavní oblast Nový prvek

Id 0

Id návrhu LCD 2

Obrázek Nový

Prezentace Power Point Nová

Graf MS Excel Nový

Tabulka MS Excel Nová

Odkaz WWW Nový

Video Nové

Návštěva Nová

Tabulka DB Nová

Data SAP Nová

Speciální - má přednost

Pořadí 5

Limit pro zobrazení Nový

Zobrazeno (sekund) 0

Uložit

Obr. 36. Přidání nové položky cyklu

12 POMOCNÁ KNIHOVNA VISUALISATIONTOOLS

V tuto chvíli je praví okamžik na představení pomocné knihovny *VisualisationTools*, která zajišťuje zobrazení vizualizačních prvků a celých cyklů na LCD panelech a v náhledech aplikace ovládání. Vytvoření společné knihovny pro vizuální prvky velmi zjednodušilo a zpřehlednilo kód. Je tím také zajištěná konzistence úprav při ladění nebo v případě požadavků na změny v průběhu života aplikace samotné.

Na základě typu vizuálního objektu a jeho ID příslušná třída *ViewModel* zajistí zobrazení (tedy čtení dat z databáze a případné uložení do dočasného adresáře) rozhraní LCD panelu v komponentě *View*.

Podobným způsobem na základě jména PC, na kterém je zobrazená aplikace *Zobrazení LCD* nebo aplikace *Zobrazení dotykového LCD*, jsou ve třídách knihovny *VisualisationTools* načtené cykly pro zobrazení a z nich se potom volají třídy pro zobrazení jednotlivých prvků zmíněné v předchozím odstavci.

12.1 Výběr dat dle časového omezení

V přílohách *LCD_s_MainCycle* a *Calendar_DayOfWeek* bych rád demonstroval výhodu použití uložených procedur MS SQL serveru. Jedná se o skalární funkci *Calendar_DayOfWeek*, jejímž účelem je vytvořit z aktuálního datumu zkratku pro aktuální den v týdnu a uloženou proceduru *LCD_s_MainCycle*, která vybírá takové položky cyklu LCD panelů, které odpovídají omezení datumu, času a dne v týdnu.

Je zřejmé, že díky takovému oddělení části výpočtů přímo do dotazu zjednodušuje zdrojový kód webové služby a také urychluje výpočet.

12.2 Třídy ViewModel

Podobně jako *aplikace ovládání*, obsahuje pomocná knihovna *Visualisation Tools* základní třídy *ViewModelBase*, *CommandViewModel* a *WorkspaceViewModel*. Třídy mají ovšem jiné vlastnosti, například pro zobrazení na LCD panelu není potřeba *DisplayName*, zato vyžadují rozhraní pro nastavení velikosti a pozice zobrazovací oblasti.

Klíčovými třídami, které dědí od *WorkspaceViewModel* a jsou také uvedené v přílohách jsou:

- *ComputerAreaContainerViewModel.cs*
- *LcdAreaContainerViewModel.cs*
- *MainAreaContainerViewModel.cs*, *AditonalAreaContainerViewModel.cs*, *Running-TextAreaContainerViewModel.cs*

Třída *ComputerAreaContainerViewModel.cs* vyvolaná pomocí vstupní třídy knihovny *CyclesReader.cs*, která obsahuje pouze volání této třídy, čte z databáze seznam LCD panelů připojených k PC, resp. generuje *LcdLayoutsVisualisationRepository*. Na základě tohoto seznamu jsou potom konstruovány jednotlivé *LcdAreaContainerViewModel.cs*, které dále na základě nastavení layoutu konstruují *MainAreaContainerViewModel.cs*, *AditonalAreaContainerViewModel.cs*, *RunningTextAreaContainerViewModel.cs*. Z jednotlivých tříd jsou potom volané konkrétní objekty pro zobrazení.

Jak je patrné z přílohy *MainAreaContainerViewModel.cs*, aktualizace vizualizačních objektů je zajištěna novým vláknem, přičemž o obnovení zobrazení v pohledu se stará *OnPropertyChanged* daného prezentačního prvku rozhraní.

Stejným způsobem je zajištěna vizualizace jak pro náhled v *aplikaci ovládání*, tak také v aplikacích LCD panelů.

13 APLIKACE ZOBRAZENÍ LCD

Obě aplikace *Zobrazeni LCD* a *zobrazení dotykového LCD* jsou ve skutečnosti velmi jednoduché programy, které se převážně spoléhají na logiku pomocné knihovny *VisualisationTools*. Prostřednictvím názvu počítače, na kterém jsou spuštěné získají cykly, nebo rozhraní dotykového displeje.

Dalším úkolem aplikací zobrazení je potom zajištění své vlastní aktualizace prostřednictvím pravidelného spouštění nástroje *AppUpdater*.

14 AUTOMATICKÁ AKTUALIZACE DAT

Automatická aktualizace je konzolová aplikace. Má na starost udržovat data v databázi aktuální pro zobrazení na jednotlivých LCD panelech. Vybrané vizualizační objekty, například soubory Excel, často upravované uživateli, mohou být určeny administrátorem v *Aplikaci ovládání* k automatické, periodicky se opakující aktualizaci. V části schématu databáze na obrázku obr. 12 je vidět, které objekty automatickou aktualizaci umožňují, jsou svázané s tabulkou *tbl_LCD_AutoUpdateList*.

- Obrázky
- Prezentace Power point
- Tabulky a grafy Excel
- Videá
- Dokumenty DOC a PDF (pro dotykový LCD panel)

Program automatické aktualizace *DataUpdateMachineWopvs.exe* je spouštěný jako naplánovaná úloha systému Windows a to maximálně 2x denně. Pokud by byla potřeba častější aktualizace, je nezbytné brát v úvahu množství aktualizovaných dat, aby zbytečně nezatěžovala systém. V takovém případě lze aktualizace rozdělit pomocí parametrů *DataUpdateMachineWopvs.exe* po jednotlivých objektech a aktualizovat pouze ty, které se mění nejčastěji, tedy naplánovat pro ně častěji se opakující naplánované úlohy. Zpravidla to bývají soubory Excel, které jsou navázané na výrobu pomocí vzorců.

ZÁVĚR

V průběhu práce na projektu jsem se seznámil s novou technologií tvorby aplikací. S použitím návrhového vzoru MVVM jsem vytvořil aplikace na ovládání a zobrazování dat na průmyslových LCD panelech ve výrobě a kancelářských prostorách. Využil jsem nové technologie poskytované moderním Frameworkem 4.5 a MS SQL 2008 R2. Jedná se o technologie od společnosti Microsoft, které jsou podporované společností ABB.

Třívrstvá architektura tvořená klientskou aplikací WPF, webovou službou WCF a uloženými procedurami v databázi MS SQL, je poměrně časově náročná na realizaci. Výhodou tohoto postupu je maximální kontrola nad akcemi, které kód provádí, neboť automaticky generovaného kódu je minimum.

Využití pravidel návrhového vzoru MVVM je v rámci tvorby kostry aplikace velmi zdoluhavé a na menší projekty, jako například *AppSaver* nebo *AppUpdater* v mém řešení, se nevyplatí. Na druhou stranu při tvorbě větších projektů, například aplikace ovládání, spatřuji výhody zejména ve velmi snadné rozšiřitelnosti a udržitelnosti kódu. Z přirozených pravidel návrhového vzoru MVVM tak vyplývá modulárnost aplikace, která byla jednou z hlavních podmínek, neboť je předpokládána rozšiřitelnost o další funkce. Návrhový vzor MVVM také poskytuje možnost provázat objekty tak, aby byl uživatel ušetřen neustálého otevírání a zavírání formulářů s číselníky a kvůli aktualizaci komboboxů.

Práce s několika webovými službami WCF a s ní související automatický generátor *Service Reference*, může být ze začátku nevyzpytatelný z hlediska jmenných prostorů a to ve smyslu použití společných tříd, fungujících na straně serveru správně, na straně klienta nejsou potom přípustné.

WCF sjednocuje dřívější webové technologie společnosti Microsoft. Díky tomu WCF pro programátora velmi zjednodušuje práci a poskytuje mnoho vlastností, kterými lze služby optimalizovat pro konkrétní případ komunikace.

SEZNAM POUŽITÉ LITERATURY

- [1] *The client/server model* [online]. [cit. 2013-5-5]. Dostupný z WWW: <http://publib.boulder.ibm.com/infocenter/txformp/v6r1/index.jsp?topic=/com.ibm.cics.tx.doc/erziaz/clientserversection.html>.
- [2] ULLMAN, Larry. *PHP a MySQL: názorný průvodce tvorbou dynamických WWW stránek*. Vyd. 1. Brno: Computer Press, 2004, 534 s. ISBN 80-251-0063-4.
- [3] *Wikipedia EN: PHP* [online]. [cit. 2013-5-5]. Dostupný z WWW: <https://en.wikipedia.org/wiki/PHP>.
- [4] *MySQL Documentation: What is MySQL* [online]. [cit. 2013-5-5]. Dostupný z WWW: <http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html>.
- [5] *Wikipedia EN: .NET Framework* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/.NET_Framework.
- [6] *Wikipedia EN: Microsoft SQL Server* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Microsoft_SQL_Server.
- [7] *Wikipedia EN: Fat client* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Fat_client.
- [8] *Wikipedia EN: Thin client* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Thin_client.
- [9] *Wikipedia EN: Windows Forms* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Windows_Forms.
- [10] *Wikipedia EN: Windows Presentation Foundation* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Windows_Presentation_Foundation.
- [11] *Wikipedia EN: Web application* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Web_application.
- [12] *Wikipedia EN: Web service* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Web_service.
- [13] *ComputerWorld: Representational State Transfer* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://www.computerworld.com/s/article/297424/Representational_State_Transfer_REST_.
- [14] *Wikipedia EN: WCF* [online]. [cit. 2013-5-5]. Dostupný z WWW: http://en.wikipedia.org/wiki/Windows_Communication_Foundation.

- [15] *Database design basics* [online]. [cit. 2013-5-6]. Dostupný z WWW: <http://office.microsoft.com/en-us/access-help/database-design-basics-HA001224247.aspx>.
- [16] *Database Design and Modeling Fundamentals* [online]. [cit. 2013-5-6]. Dostupný z WWW: <http://www.sqlteam.com/article/database-design-and-modeling-fundamentals>.
- [17] CONOLLY, Thomas, Carolyn E BEGG a Richard HOLOWCZAK. *Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází*. Vyd. 1. Brno: Computer Press, 2009, 584 s. ISBN 978-80-251-2328-7.
- [18] RIORDAN, Rebecca M, Carolyn E BEGG a Richard HOLOWCZAK. *Vytváříme relační databázové aplikace: profesionální průvodce tvorbou efektivních databází*. Vyd. 1. Praha: Computer Press, 2000, 584 s. ISBN 80-722-6360-9.
- [19] *LocBaml Tool Sample* [online]. [cit. 2013-5-6]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms771568.aspx>.
- [20] SMITH-FERRIER, Guy. *.NET internationalization: the developer's guide to building global Windows and Web applications*. Upper Saddle River, NJ: Addison-Wesley, c2007, xxxi, 639 p. ISBN 978-032-1341-389.
- [21] STRAHL, Rick and Bustamante Michele Leroux. *WPF Localization Guidance*. Upper Saddle River, NJ: Addison-Wesley, c2007, xxxi, 639 p. ISBN 978-032-1341-389.
- [22] *WPF Apps With The Model-View-ViewModel Design Pattern* [online]. [cit. 2013-5-6]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>.
- [23] LAIR, Robert. *Beginning Silverlight 5 in C#. 4th ed.* New York: Apress. ISBN 978-143-0234-623.
- [24] GAROFALO, Raffaele. *Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern*. Sebastopol, Calif: O'Reilly Media. ISBN 978-073-5650-923.
- [25] MACDONALD, Matthew a Fabio Claudio Ferracchiati TECHNICAL REVIEWER. *Pro Silverlight 5 in C?, fourth edition. 4th ed.* New York: Apress. ISBN 978-143-0234-807.
- [26] MACDONALD, Matthew. *Pro WPF in C# 2012: Windows Presentation Foundation with .NET 4.5. 4th ed.* London: Springer [distributor], 2012, pages. ISBN 14-302-4365-1.

-
- [27] HALL, Gary McLean. *Pro WPF and Silverlight MVVM: effective application development with Model-View-ViewModel*. New York: Distributed to the book trade worldwide by Springer Science Business Media, c2010, xiii, 257 p. Expert's voice in WPF. ISBN 14-302-3162-9.
- [28] HUNDHAUSEN, Richard. *Professional scrum development with microsoft visual studio 2012*. Redmond, WA: Microsoft Press, 2012, p. cm. ISBN 978-073-5657-984.
- [29] JOHNSON, Bruce Ovid. *Professional visual studio 2012. 1st ed.* Indianapolis, IN: Wiley Pub., Inc., 2012, p. cm. ISBN 11-183-3770-0.
- [30] ANDRADE, Chris. *Professional WPF programming*. Indianapolis, IN: Wiley Pub., c2007. ISBN 04-700-4180-3.
- [31] SHARP, John. *Windows Communication Foundation 4 step by step*. Sebastopol, Calif.: Microsoft/O'Reilly, c2010, xxix, 700 p. ISBN 07-356-4556-6.
- [32] Banks, Richard. *Visual Studio 2012 Cookbook*. Livery Place 35 Livery Street Birmingham B3 2PB ISBN 978-1-84968-652-5.
- [33] *Avocent LongView IP*. Avocent Corporation, 2008. Dostupné z: www.avocent.com.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MVVM	návrhový vzor Model View View-Model
PHP	Personal Home Pahe
CLR	Common Language Runtime
API	Application Programming Interface
MFC	Microsoft Foundation Class Library
WPF	Windows Presentation Foundation
WSDL	Web Services Description Language
SOAP	Simple Object Acces Protocol
REST	Representative State Transfer
WCF	Windows Communication Foundation
XAML	Extensible Application Markup Language
MVC	Model View Controller
MVP	Model View Presenter
SAP	Systems, Applications, Products in data processing

SEZNAM OBRÁZKŮ

Obr. 1. Možnosti zapojení datového převodník LondView IP[33]	28
Obr. 2. Přehled funkcí aplikace ovládání LCD panelů	36
Obr. 3. Příklad LCD panelů připojených k PC.....	37
Obr. 4. Hlavní nabídka dotykového LCD.....	37
Obr. 5. Podnabídka Company dotykového LCD	38
Obr. 6. Podnabídka Products portfolio dotykového LCD	38
Obr. 7. Podnabídka Producion dotykového LCD	39
Obr. 8. Podnabídka Other dotykového LCD	39
Obr. 9. Přehled funkcí aplikace ovládání LCD panelů	40
Obr. 10. Příklad struktury zařízení na síti.....	41
Obr. 11. Návrh DB : Tabulky pro uložení částí aplikace.....	44
Obr. 12. Návrh DB : Celkový návrh cyklů LCD panelů	47
Obr. 13. Návrh DB : Cykly LCD panelů : Rozvržení LCD panelu a PC.....	47
Obr. 14. Návrh DB : Cykly LCD panelů : Časová omezení cyklů	48
Obr. 15. Návrh DB : Detail části cyklů LCD panelů.....	48
Obr. 16. Návrh DB : Nabídky a obsah dotykových panelů.....	49
Obr. 17. Návrh DB : Detail nabídky a příklad obsahu dotykových panelů.....	50
Obr. 18. Návrh DB : Autorizace pro aplikaci ovládání	50
Obr. 19. Ukládání nových verzí klientské aplikace	54
Obr. 20. Ukládání nových verzí klientské aplikace	55
Obr. 21. Čtení nových verzí klientské aplikace	56
Obr. 22. Hierarchie tříd ViewModel.....	58
Obr. 23. Nabídka Soubor / Funkce administrátora	60
Obr. 24. Nabídka Soubor / Jazyk	61
Obr. 25. Nabídka Přidat/odebrat obsah.....	62
Obr. 26. Nabídka Nastavení LCD.....	63
Obr. 27. Nabídka Nastavení dotykového LCD.....	64
Obr. 28. Seznam obrázků	64
Obr. 29. Formulář pro vložení nového obrázku do databáze.....	65
Obr. 30. Příklad zobrazení obrázku v aplikaci ovládání.....	66
Obr. 31. Přehled vytvořených rozvržení LCD	67
Obr. 32. Návrh nového rozvržení LCD.....	69
Obr. 33. Doplnění nového ovládacího PC k rozvržení	70
Obr. 34. Přehled rozvržení LCD panelů	70
Obr. 35. Hlavní cykly LCD panelu	71
Obr. 36. Přidání nové položky cyklu.....	72

SEZNAM TABULEK

Tab. 1. Zobrazované typy dat 34

SEZNAM PŘÍLOH

- P I. Web.config
- P II. DBdataExchange.cs
- P III. IControlInterfaceService.cs
- P IV. ControlInterfaceService.svc
- P V. IFileTransferService.cs
- P VI. FileTransferService.svc
- P VII. RelayCommand.cs
- P VIII. ViewModelBase.cs
- P IX. CommandViewModel.cs
- P X. MainWindowViewModel.cs
- P XI. MainWindowWorkspacesViewModel.cs
- P XII. VisuBaseObject.cs
- P XIII. LCD_s_MainCycle
- P XIV. Calendar_DayOfWeek
- P XV. ComputerAreaContainerViewModel
- P XVI. LcdAreaContainerViewModel
- P XVII. MainAreaContainerViewModel

PŘÍLOHA P I. WEB.CONFIG

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext"
      value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime maxRequestLength="2097151" />
  </system.web>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IFileTransferService"
          transferMode="StreamedRequest"
          maxBufferSize="65536"
          maxReceivedMessageSize="2147483647"
          messageEncoding="Mtom"
          receiveTimeout="00:10:00">
          <readerQuotas maxDepth="128"
            maxStringContentLength="2147483647"
            maxArrayLength="2147483647"
            maxBytesPerRead="2147483647"
            maxNameTableCharCount="2147483647" />
          <security mode="None">
            <transport clientCredentialType="None"
              proxyCredentialType="None" realm="" />
            <message clientCredentialType="UserName"
              algorithmSuite="Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <services>
      <service behaviorConfiguration="FileTransferServiceBehavior"
        name="WsWopvs.FileTransferService">
        <endpoint address=""
          binding="basicHttpBinding"
          bindingConfiguration="
            BasicHttpBinding_IFileTransferService"
          contract="WsWopvs.IFileTransferService"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="FileTransferServiceBehavior">
          <serviceMetadata httpGetEnabled="true"/>
          <dataContractSerializer maxItemsInObjectGraph="2147483647" />
          <serviceThrottling
            maxConcurrentCalls="500"
            maxConcurrentSessions="500"
            maxConcurrentInstances="500"
          />
        </behavior>
        <behavior>
          <!--To avoid disclosing metadata information, set the values
            below to false before deployment-->
          <serviceMetadata httpGetEnabled="true"/>
          <!--To receive exception details in faults for debugging
            purposes, set the value below to true. Set to false
            before deployment to avoid disclosing exception
            information-->
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

```
        </behavior>
    </serviceBehaviors>
</behaviors>
<protocolMapping>
    <add binding="basicHttpBinding" scheme="http" />
</protocolMapping>
<serviceHostingEnvironment aspNetCompatibilityEnabled="false"
    multipleSiteBindingsEnabled="true"
    minFreeMemoryPercentageToActivateService="0" />
</system.serviceModel>
<system.webServer>
    <security>
        <requestFiltering>
            <requestLimits maxAllowedContentLength="2147483647"/>
        </requestFiltering>
    </security>
    <modules runAllManagedModulesForAllRequests="true"/>
    <!--
        To browse web app root directory during debugging,
        set the value below to true. Set to false before deployment to
        avoid disclosing web app folder information.
    -->
    <directoryBrowse enabled="true"/>
</system.webServer>
</configuration>
```

PŘÍLOHA P II. DBDATAEXCHANGE.CS

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
namespace WsWopvs
{
    public class DBdataExchange
    {
        public DBdataExchange(int userID, string procName)
        {
            this.userID = userID;
            this.procName = procName;
        }
        private int userID;
        private string procName;
        public DataTable EmptyTable
        {
            get
            {
                return new DataTable("dtEmpty");
            }
        }
        public DataSet DataExchange(DataTable dtParameters)
        {
            DataSet dsFncRet = new DataSet("dsFncRet");
            try
            {
                if (dtParameters.Rows.Count == 0)
                {
                    dsFncRet.Merge(CallProcedure());
                }
                else
                {
                    foreach (DataRow dr in dtParameters.Rows)
                    {
                        dsFncRet.Merge(CallProcedure(SqlParamatersFromDataRow(dr)));
                    }
                }
                WriteLog();
            }
            catch (Exception Exc)
            {
                throw Exc;
            }
            return dsFncRet;
        }
        private DataSet CallProcedure()
        {
            SqlConnection connSql
                = new SqlConnection(Constants.ConnectionStringDB);
            try
            {
                SqlCommand cmdSql = new SqlCommand(procName, connSql);
                cmdSql.CommandType = CommandType.StoredProcedure;
                SqlDataAdapter adapter = new SqlDataAdapter(cmdSql);
                DataSet ds = new DataSet();
                adapter.Fill(ds);
                return ds;
            }
            catch (Exception Exc)
```

```

    {
        Errors e = new Errors(Exc);
        throw Exc;
    }
    finally
    {
        connSql.Close();
    }
}
private DataSet CallProcedure(SqlParameter[] parameters)
{
    SqlConnection connSql
        = new SqlConnection(Constants.ConnectionStringDB);
    try
    {
        SqlCommand cmdSql = new SqlCommand(procName, connSql);
        cmdSql.CommandType = CommandType.StoredProcedure;
        foreach (SqlParameter parameter in parameters)
        {
            if (parameter.Value == null)
                parameter.Value = DBNull.Value;
            cmdSql.Parameters.AddWithValue(parameter.ParameterName,
                parameter.Value);
        }
        SqlDataAdapter adapter = new SqlDataAdapter(cmdSql);
        DataSet ds = new DataSet();
        adapter.Fill(ds);
        return ds;
    }
    catch (Exception Exc)
    {
        Errors e = new Errors(Exc);
        throw Exc;
    }
    finally
    {
        connSql.Close();
    }
}
private SqlParameter[] GetParameters()
{
    SqlConnection connSql
        = new SqlConnection(Constants.ConnectionStringDB);
    try
    {
        connSql.Open();
        SqlCommand cmdSql = new SqlCommand(procName, connSql);
        cmdSql.CommandType = CommandType.StoredProcedure;
        SqlCommandBuilder.DeriveParameters(cmdSql);
        cmdSql.Parameters.RemoveAt(0);
        SqlParameter[] parameters
            = new SqlParameter[cmdSql.Parameters.Count];
        cmdSql.Parameters.CopyTo(parameters, 0);
        return parameters;
    }
    catch (Exception Exc)
    {
        Errors e = new Errors(Exc);
        throw Exc;
    }
    finally
    {
        connSql.Close();
    }
}

```


PŘÍLOHA P III. ICONTROLINTERFACESERVICE.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WsWopvs
{
    [ServiceContract]
    public interface IControlInterfaceService
    {
        [OperationContract]
        List<VisuObject> VisuObjectList(VisuObjectKind objectKind,
            int userID);
        [OperationContract]
        void VisuObjectsToSave(VisuObjectKind objectKind,
            List<VisuObject> visuObjects, int userID);
        [OperationContract]
        void VisuObjectsToDelete(VisuObjectKind objectKind,
            List<VisuObject> visuObjects, int userID);
        [OperationContract]
        int SaveOneVisuObject(VisuObjectKind objectKind,
            VisuObject visuObject, int userID);
        ...
    }
    #region Lcd setting
    #endregion
    #region Visualisation objects
    public enum VisuObjectKind
    {
        Nothing = 0,
        Picture = 1,
        Presentation = 2,
        ExcelGraph = 3,
        ExcelTable = 4,
        WwwLink = 5,
        Video = 6,
        Visitor = 7,
        TextInfo = 8,
        SapData = 9,
        DocumentDocPdf = 10
    }
    public class VisuObject
    {
        public int Id;
        public string FileName;
        public string FilePath;
        public DateTime CreatedAt;
        public string AuthorName;
        public string LastEditorName;
        public bool AutoUpdate;
    }
    #endregion
}
```

PŘÍLOHA P IV. CONTROLINTERFACESERVICE.SVC

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WsWopvs
{
    public class ControlInterfaceService : IControlInterfaceService
    {
        #region Visualisation objects
        public List<VisuObject> VisuObjectList(VisuObjectKind objectKind,
            int userID)
        {
            string procName;
            switch (objectKind)
            {
                case VisuObjectKind.Picture:
                    procName = "LCD_CI_s_PictureList";
                    break;
                case VisuObjectKind.Presentation:
                    procName = "LCD_CI_s_PresentationList";
                    break;
                default:
                    throw new Exception("Reading the Visualisation Objects
                    the Object Kind is undefined! Object name: "
                    + objectKind.ToString());
            }
            try
            {
                DBdataExchange de = new DBdataExchange(userID, procName);
                DataSet ds = de.DataExchange(de.EmptyTable);
                List<VisuObject> FncRet = new List<VisuObject>();
                FncRet = (from DataRow dr in ds.Tables[0].Rows
                    select new VisuObject
                    {
                        Id = (int)dr["ID"],
                        FileName = dr["FileName"].ToString(),
                        FilePath = dr["FilePath"].ToString(),
                        CreatedAt = (DateTime)dr["CreatedAt"],
                        AuthorName = dr["AuthorName"].ToString(),
                        LastEditorName = dr["LastEditorName"].ToString(),
                        AutoUpdate = (bool)dr["AutoUpdate"]
                    }).ToList();
                return FncRet;
            }
            catch (Exception Exc)
            {
                throw Exc;
            }
        }
        public void VisuObjectsToSave(VisuObjectKind objectKind,
            List<VisuObject> visuObjects, int userID)
        {
            string procName;
            DataTable dt = new DataTable("VisuObjects");
            switch (objectKind)
            {
                case VisuObjectKind.Picture:
                    procName = "LCD_CI_iu_Picture";
```

```

        dt.Columns.Add("ID", Type.GetType("System.Int32"));
        dt.Columns.Add("PictureName", Type.GetType("System.String"));
        dt.Columns.Add("PicturePath", Type.GetType("System.String"));
        dt.Columns.Add("CreatedAt", Type.GetType("System.DateTime"));
        dt.Columns.Add("AuthorId", Type.GetType("System.Int32"));
        dt.Columns.Add("LastEditorId", Type.GetType("System.Int32"));
        dt.Columns.Add("AutoUpdate", Type.GetType("System.Boolean"));
        break;
    case VisuObjectKind.Presentation:
        procName = "LCD_CI_iu_Presentation";
        dt.Columns.Add("ID", Type.GetType("System.Int32"));
        dt.Columns.Add("PptName", Type.GetType("System.String"));
        dt.Columns.Add("PptPath", Type.GetType("System.String"));
        dt.Columns.Add("CreatedAt", Type.GetType("System.DateTime"));
        dt.Columns.Add("AuthorId", Type.GetType("System.Int32"));
        dt.Columns.Add("LastEditorId", Type.GetType("System.Int32"));
        dt.Columns.Add("AutoUpdate", Type.GetType("System.Boolean"));
        break;
    default:
        throw new Exception("Writing the Visualisation Objects the
            Object Kind is undefined! Object name: "
            + objectKind.ToString());
}
try
{
    DBdataExchange de = new DBdataExchange(userID, procName);
    DataRow dr;
    switch (objectKind)
    {
        case VisuObjectKind.Picture:
        case VisuObjectKind.Presentation:
            foreach (VisuObject vo in visuObjects)
            {
                dr = dt.NewRow();
                dr[0] = vo.Id;
                dr[1] = vo.FileName;
                dr[2] = vo.FilePath;
                dr[3] = vo.CreatedAt;
                if ((vo.AuthorName == "") ||
                    (vo.AuthorName == null))
                    dr[4] = userID;
                else
                    dr[5] = userID;
                dr[6] = vo.AutoUpdate;
                dt.Rows.Add(dr);
            }
            break;
    }
    de.DataExchange(dt);
}
catch (Exception Exc)
{
    throw Exc;
}
}
public void VisuObjectsToDelete(VisuObjectKind objectKind,
    List<VisuObject> visuObjects, int userID)
{
    string procName;
    switch (objectKind)
    {
        case VisuObjectKind.Picture:
            procName = "LCD_CI_d_Picture";

```

```

        break;
    case VisuObjectKind.Presentation:
        procName = "LCD_CI_d_Presentation";
        break;
    default:
        throw new Exception("Deleting the Visualisation Objects
the Object Kind is undefined! Object name: "
+ objectKind.ToString());
}
try
{
    DBdataExchange de = new DBdataExchange(userID, procName);
    DataTable dt = new DataTable("VisuObjects");
    dt.Columns.Add("ID", Type.GetType("System.Int32"));
    DataRow dr;
    foreach (VisuObject vo in visuObjects)
    {
        dr = dt.NewRow();
        dr[0] = vo.Id;
        dt.Rows.Add(dr);
    }
    de.DataExchange(dt);
}
catch (Exception Exc)
{
    throw Exc;
}
}
public int SaveOneVisuObject(VisuObjectKind objectKind,
    VisuObject visuObject, int userID)
{
    int FncRet = 0;
    string procName;
    DataTable dt = new DataTable("VisuObjects");
    switch (objectKind)
    {
        case VisuObjectKind.Picture:
            procName = "LCD_CI_i_Picture";
            dt.Columns.Add("ID", Type.GetType("System.Int32"));
            dt.Columns.Add("PictureName", Type.GetType("System.String"));
            dt.Columns.Add("PicturePath", Type.GetType("System.String"));
            dt.Columns.Add("CreatedAt", Type.GetType("System.DateTime"));
            dt.Columns.Add("AuthorId", Type.GetType("System.Int32"));
            dt.Columns.Add("LastEditorId", Type.GetType("System.Int32"));
            dt.Columns.Add("AutoUpdate", Type.GetType("System.Boolean"));
            break;
        case VisuObjectKind.Presentation:
            procName = "LCD_CI_i_Presentation";
            dt.Columns.Add("ID", Type.GetType("System.Int32"));
            dt.Columns.Add("PptName", Type.GetType("System.String"));
            dt.Columns.Add("PptPath", Type.GetType("System.String"));
            dt.Columns.Add("CreatedAt", Type.GetType("System.DateTime"));
            dt.Columns.Add("AuthorId", Type.GetType("System.Int32"));
            dt.Columns.Add("LastEditorId", Type.GetType("System.Int32"));
            dt.Columns.Add("AutoUpdate", Type.GetType("System.Boolean"));
            break;
        default:
            throw new Exception("Writing the Visualisation Objects the
Object Kind is undefined! Object name: "
+ objectKind.ToString());
    }
}
try
{

```

```

DBDataExchange de = new DBDataExchange(userID, procName);
DataRow dr;
switch (objectKind)
{
    case VisuObjectKind.Picture:
    case VisuObjectKind.Presentation:
        dr = dt.NewRow();
        dr[0] = visuObject.Id;
        dr[1] = visuObject.FileName;
        dr[2] = visuObject.FilePath;
        dr[3] = visuObject.CreatedAt;
        if ((visuObject.AuthorName == "") ||
            (visuObject.AuthorName == null))
            dr[4] = userID;
        else
            dr[5] = userID;
        dr[6] = visuObject.AutoUpdate;
        dt.Rows.Add(dr);
        break;
}
FncRet = (int)de.DataExchange(dt).Tables[0].Rows[0][0];
}
catch (Exception Exc)
{
    throw Exc;
}
return FncRet;
}
#endregion
...
}
}

```

PŘÍLOHA P V. IFILETRANSFERSERVICE.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WsWopvs
{
    [ServiceContract]
    public interface IFileTransferService
    {
        [OperationContract]
        void AppFileDataUpload(SendFileRequestMessage request);
        [OperationContract]
        DownloadFileInfo
        AppFileDataDownload(DownloadFileTransferRequest info);
        [OperationContract]
        FTVisuObjectKind TestObj(FTVisuObjectKind objKind);
        [OperationContract]
        void VisuFileDataUpload(SendVisuFileRequestMessage request);
        [OperationContract]
        DownloadVisuFileInfo
        VisuFileDataDownload(DownloadVisuFileTransferRequest info);
    }
    #region Application files upload download
    // upload part
    [MessageContract]
    public class SendFileRequestMessage : IDisposable
    {
        [MessageHeader(MustUnderstand = true)]
        public FileTransferInfo FileInfo;
        [MessageBodyMember(Order = 1)]
        public System.IO.Stream FileByteStream;
        public void Dispose()
        {
            if (FileByteStream != null)
            {
                FileByteStream.Close();
                FileByteStream = null;
            }
        }
    }
    [DataContract]
    public class FileTransferInfo
    {
        [DataMember(Order = 1, IsRequired = true)]
        public ApplicationsWopvs App;
        [DataMember(Order = 2, IsRequired = true)]
        public int FileId;
        [DataMember(Order = 3, IsRequired = true)]
        public int UserId;
        [DataMember(Order = 4, IsRequired = true)]
        public long Checksum;
    }
    // download part
    [MessageContract]
    public class DownloadFileTransferRequest
    {
        [MessageBodyMember]
        public ApplicationsWopvs App;
    }
}

```

```

        [MessageBodyMember]
        public int FileId;
        [MessageBodyMember]
        public int UserId;
    }
    [MessageContract]
    public class DownloadFileInfo : IDisposable
    {
        [MessageHeader(MustUnderstand = true)]
        public string FileName;
        [MessageBodyMember(Order = 1)]
        public System.IO.Stream FileByteStream;
        public void Dispose()
        {
            if (FileByteStream != null)
            {
                FileByteStream.Close();
                FileByteStream = null;
            }
        }
    }
}
#endregion
#region Visualisation objects files upload download
public enum FTVisuObjectKind
{
    Nothing = 0,
    Picture = 1,
    Presentation = 2,
    ExcelGraph = 3,
    ExcelTable = 4,
    WwwLink = 5,
    Video = 6,
    Visitor = 7,
    TextInfo = 8,
    SapData = 9,
    DocumentDocPdf = 10
}
// upload part
[DataContract]
public class VisuFileTransferInfo
{
    [DataMember(Order = 1, IsRequired = true)]
    public FTVisuObjectKind VisuObj;
    [DataMember(Order = 2, IsRequired = true)]
    public int VisuFileId;
    [DataMember(Order = 3, IsRequired = true)]
    public int UserId;
    [DataMember(Order = 4, IsRequired = true)]
    public long Checksum;
}
[MessageContract]
public class SendVisuFileRequestMessage : IDisposable
{
    [MessageHeader(MustUnderstand = true)]
    public VisuFileTransferInfo VisuFileInfo;
    [MessageBodyMember(Order = 1)]
    public System.IO.Stream VisuFileByteStream;
    public void Dispose()
    {
        if (VisuFileByteStream != null)
        {
            VisuFileByteStream.Close();
        }
    }
}

```

```

        VisuFileByteStream = null;
    }
}
// download part
[MessageContract]
public class DownloadVisuFileTransferRequest
{
    [MessageBodyMember]
    public FTVisuObjectKind VisuObj;
    [MessageBodyMember]
    public int VisuFileId;
    [MessageBodyMember]
    public int UserId;
}
[MessageContract]
public class DownloadVisuFileInfo : IDisposable
{
    [MessageHeader(MustUnderstand = true)]
    public string VisuFileName;
    [MessageBodyMember(Order = 1)]
    public System.IO.Stream VisuFileByteStream;
    public void Dispose()
    {
        if (VisuFileByteStream != null)
        {
            VisuFileByteStream.Close();
            VisuFileByteStream = null;
        }
    }
}
}
#endregion
}

```

PŘÍLOHA P VI. FILETRANSFERSERVICE.SVC

```
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WsWopvs
{
    public class FileTransferService : IFileTransferService
    {
        #region Application files upload download
        public void AppFileDataUpload(SendFileRequestMessage request)
        {
            Stream sourceStream = request.FileByteStream;
            const int bufferLen = 65000;
            byte[] buffer = new byte[bufferLen];
            MemoryStream ms = new MemoryStream();
            int bytesRead, totalBytesRead = 0;
            do
            {
                bytesRead = sourceStream.Read(buffer, 0, buffer.Length);
                totalBytesRead += bytesRead;
                ms.Write(buffer, 0, bytesRead);
            } while (bytesRead > 0);
            // Save the file on database.
            string procName;
            switch (request.FileInfo.App)
            {
                case ApplicationsWopvs.ControlInterface:
                    procName = "App_u_ControlInterfaceFileUpload";
                    break;
                case ApplicationsWopvs.DataUpdateMachine:
                    procName = "App_u_DataUpdateMachineFileUpload";
                    break;
                case ApplicationsWopvs.LcdInterface:
                    procName = "App_u_LcdInterfaceFileUpload";
                    break;
                case ApplicationsWopvs.TouchLcdInterface:
                    procName = "App_u_TouchLcdInterfaceFileUpload";
                    break;
                default:
                    throw new Exception("Application to udate undefined");
            }
        }
        try
        {
            DBdataExchange de
                = new DBdataExchange(request.FileInfo.UserId, procName);
            DataTable dt = new DataTable("AppFiles");
            dt.Columns.Add("ID", Type.GetType("System.Int32"));
            dt.Columns.Add("DataOfFile", Type.GetType("System.Byte[]"));
            DataRow dr;
            dr = dt.NewRow();
            dr[0] = request.FileInfo.FileId;
            dr[1] = ms.ToArray();
            dt.Rows.Add(dr);
            de.DataExchange(dt);
        }
        catch (Exception Exc)
        {

```

```

        throw Exc;
    }
    finally
    {
        ms.Close();
    }
}
public DownloadFileInfo
AppFileDataDownload(DownloadFileTransferRequest info)
{
    string procName;
    MemoryStream ms = null;
    DownloadFileInfo df = new DownloadFileInfo();
    switch (info.App)
    {
        case ApplicationsWopvs.ControlInterface:
            procName = "App_s_ControlInterfaceFileData";
            break;
        case ApplicationsWopvs.DataUpdateMachine:
            procName = "App_s_DataUpdateMachineFileData";
            break;
        case ApplicationsWopvs.LcdInterface:
            procName = "App_s_LcdInterfaceFileData";
            break;
        case ApplicationsWopvs.TouchLcdInterface:
            procName = "App_s_TouchLcdInterfaceFileData";
            break;
        default:
            throw new Exception("Application file to
                download undefined");
    }
    try
    {
        DBdataExchange de = new DBdataExchange(info.UserId,
            procName);
        DataTable dt = new DataTable("AppFiles");
        dt.Columns.Add("ID", Type.GetType("System.Int32"));
        DataRow dr;
        dr = dt.NewRow();
        dr[0] = info.FileId;
        dt.Rows.Add(dr);
        DataSet dsTmp = de.DataExchange(dt);
        ms = new MemoryStream((byte[])dsTmp.Tables[0].Rows[0][0]);
        df.FileName = dsTmp.Tables[0].Rows[0][1].ToString();
        df.FileByteStream = ms;
    }
    catch (Exception Exc)
    {
        throw Exc;
    }
    return df;
}
#endregion
#region Visualisation objects files upload download
public FTVisuObjectKind TestObj(FTVisuObjectKind objKind)
{
    return objKind;
}
public void
VisuFileDataUpload(SendVisuFileRequestMessage request)
{
    Stream sourceStream = request.VisuFileByteStream;
    const int bufferLen = 65000;
}

```

```

byte[] buffer = new byte[bufferLen];
MemoryStream ms = new MemoryStream();
int bytesRead, totalBytesRead = 0;
do
{
    bytesRead = sourceStream.Read(buffer, 0, buffer.Length);
    totalBytesRead += bytesRead;
    ms.Write(buffer, 0, bytesRead);
} while (bytesRead > 0);
// Save the file on database.
string procName;
switch (request.VisuFileInfo.VisuObj)
{
    case FTVisuObjectKind.Picture:
        procName = "LCD_CI_u_PictureFileUpload";
        break;
    case FTVisuObjectKind.Presentation:
        procName = "LCD_CI_u_PresentationFileUpload";
        break;
    default:
        throw new Exception("Visualisation file to
            udate undefined");
}
try
{
    DBdataExchange de
        = new DBdataExchange(request.VisuFileInfo.UserId,
            procName);
    DataTable dt = new DataTable("VisuFiles");
    dt.Columns.Add("ID", Type.GetType("System.Int32"));
    dt.Columns.Add("DataOfFile", Type.GetType("System.Byte[]"));
    DataRow dr;
    dr = dt.NewRow();
    dr[0] = request.VisuFileInfo.VisuFileId;
    dr[1] = ms.ToArray();
    dt.Rows.Add(dr);
    de.DataExchange(dt);
}
catch (Exception Exc)
{
    throw Exc;
}
finally
{
    ms.Close();
}
}
public DownloadVisuFileInfo
    VisuFileDataDownload(DownloadVisuFileTransferRequest info)
{
    string procName;
    MemoryStream ms = null;
    DownloadVisuFileInfo df = new DownloadVisuFileInfo();
    switch (info.VisuObj)
    {
        case FTVisuObjectKind.Picture:
            procName = "LCD_s_PictureFileData";
            break;
        case FTVisuObjectKind.Presentation:
            procName = "LCD_s_PresentationFileData";
            break;
        default:
            throw new Exception("Visualisation object file to
                udate undefined");
    }
}

```

```
        download undefined");
    }
    try
    {
        DBdataExchange de = new DBdataExchange(info.UserId,
            procName);
        DataTable dt = new DataTable("VisuFiles");
        dt.Columns.Add("ID", Type.GetType("System.Int32"));
        DataRow dr;
        dr = dt.NewRow();
        dr[0] = info.VisuFileId;
        dt.Rows.Add(dr);
        DataSet dsTmp = de.DataExchange(dt);
        ms = new MemoryStream((byte[])dsTmp.Tables[0].Rows[0][0]);
        df.VisuFileName = dsTmp.Tables[0].Rows[0][1].ToString();
        df.VisuFileByteStream = ms;
    }
    catch (Exception Exc)
    {
        throw Exc;
    }
    return df;
}
}
#endregion
}
}
```

PŘÍLOHA P VII. RELAYCOMMAND.CS

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
namespace ControlInterfaceWopvs
{
    /// <summary>
    /// A command whose sole purpose is to
    /// relay its functionality to other
    /// objects by invoking delegates. The
    /// default return value for the CanExecute
    /// method is 'true'.
    /// </summary>
    public class RelayCommand : ICommand
    {
        #region Fields
        readonly Action<object> _execute;
        readonly Predicate<object> _canExecute;
        #endregion // Fields
        #region Constructors
        /// <summary>
        /// Creates a new command that can always execute.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        public RelayCommand(Action<object> execute)
            : this(execute, null)
        {
        }
        /// <summary>
        /// Creates a new command.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        /// <param name="canExecute">The execution status logic.</param>
        public RelayCommand(Action<object> execute,
            Predicate<object> canExecute)
        {
            if (execute == null)
                throw new ArgumentNullException("execute");
            _execute = execute;
            _canExecute = canExecute;
        }
        #endregion // Constructors
        #region ICommand Members
        [DebuggerStepThrough]
        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute(parameter);
        }
        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }
        public void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
}
```

```
}      } #endregion // ICommand Members  
}
```

PŘÍLOHA P VIII. VIEWMODELBASE.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ControlInterfaceWopvs.ViewModels
{
    /// <summary>
    /// Base class for all ViewModel classes in the application.
    /// It provides support for property change notifications
    /// and has a DisplayName property. This class is abstract.
    /// </summary>
    public abstract class ViewModelBase : INotifyPropertyChanged,
        IDisposable
    {
        #region Constructor
        protected ViewModelBase()
        {
        }
        #endregion // Constructor
        #region DisplayName
        /// <summary>
        /// Returns the user-friendly name of this object.
        /// Child classes can set this property to a new value,
        /// or override it to determine the value on-demand.
        /// </summary>
        public virtual string DisplayName { get; protected set; }
        #endregion // DisplayName
        #region Debugging Aides
        /// <summary>
        /// Warns the developer if this object does not have
        /// a public property with the specified name. This
        /// method does not exist in a Release build.
        /// </summary>
        [Conditional("DEBUG")]
        [DebuggerStepThrough]
        public void VerifyPropertyName(string propertyName)
        {
            // Verify that the property name matches a real,
            // public, instance property on this object.
            if (TypeDescriptor.GetProperties(this)[propertyName] == null)
            {
                string msg = "Invalid property name: " + propertyName;
                if (this.ThrowOnInvalidPropertyName)
                    throw new Exception(msg);
                else
                    Debug.Fail(msg);
            }
        }
        /// <summary>
        /// Returns whether an exception is thrown, or if a Debug.Fail()
        /// is used when an invalid property name is passed to the
        /// VerifyPropertyName method. The default value is false, but
        /// subclasses used by unit tests might override this property's
        /// getter to return true.
        /// </summary>
        protected virtual bool ThrowOnInvalidPropertyName
```

```

    { get; private set; }
#endregion // Debugging Aides
#region INotifyPropertyChanged Members
/// <summary>
/// Raised when a property on this object has a new value.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;
/// <summary>
/// Raises this object's PropertyChanged event.
/// </summary>
/// <param name="propertyName">The prop. that has a new val.</param>
protected virtual void OnPropertyChanged(string propertyName)
{
    this.VerifyPropertyName(propertyName);
    PropertyChangedEventHandler handler = this.PropertyChanged;
    if (handler != null)
    {
        var e = new PropertyChangedEventArgs(propertyName);
        handler(this, e);
    }
}
#endregion // INotifyPropertyChanged Members
#region IDisposable Members
/// <summary>
/// Invoked when this object is being removed from the application
/// and will be subject to garbage collection.
/// </summary>
public void Dispose()
{
    this.OnDispose();
}
/// <summary>
/// Child classes can override this method to perform
/// clean-up logic, such as removing event handlers.
/// </summary>
protected virtual void OnDispose()
{
}
}
#if DEBUG
/// <summary>
/// Useful for ensuring that ViewModel objects are properly
/// garbage collected.
/// </summary>
~ViewModelBase()
{
    string msg = string.Format("{0} ({1}) ({2}) Finalized",
        this.GetType().Name, this.DisplayName, this.GetHashCode());
    System.Diagnostics.Debug.WriteLine(msg);
}
#endif
#endregion // IDisposable Members
}
}

```

PŘÍLOHA P IX. COMMANDVIEWMODEL.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
namespace ControlInterfaceWopvs.ViewModels
{
    /// <summary>
    /// Represents an actionable item displayed by a View.
    /// </summary>
    public class CommandViewModel : ViewModelBase
    {
        public CommandViewModel(string displayName, ICommand command)
        {
            if (command == null)
                throw new ArgumentNullException("command");
            base.DisplayName = displayName;
            this.Command = command;
        }
        public ICommand Command { get; private set; }
    }
}
```

PŘÍLOHA P X. MAINWINDOWVIEWMODEL.CS

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;
using System.Windows.Input;
using ControlInterfaceWopvs.DataAccess;
using ControlInterfaceWopvs.Models;
namespace ControlInterfaceWopvs.ViewModels
{
    /// <summary>
    /// The ViewModel for the application's main window.
    /// </summary>
    public class MainWindowViewModel : WorkspaceViewModel
    {
        #region Fields
        MainWindowWorkspacesViewModel _allWorkspaces;
        private bool _optionsEnabled;
        private bool _adminOptionsEnabled;
        public int _userId;
        #endregion // Fields
        #region Constructor
        public MainWindowViewModel(string userName)
        {
            _allWorkspaces = new MainWindowWorkspacesViewModel();
            base.DisplayName = Properties.Resources.MainWindow_Name;
            AccessRights ar
                = UserAccessRightsReader.LoadUserAccessRights(userName);
            _optionsEnabled = ar.UserAuthorized;
            _userId = ar.UserId;
            _adminOptionsEnabled = ar.UserRights.Contains(1);
        }
        #endregion // Constructor
        #region Main Window Presentation Properties
        public bool OptionsEnabled { get { return _optionsEnabled; } }
        public bool AdminOptionsEnabled
            { get { return _adminOptionsEnabled; } }
        public string FileMenu
            { get { return Properties.Resources.MainWindow_File; } }
        public string LanguageSetting
            { get { return Properties.Resources.MainWindow_Language; } }
        public string LanguageCzech
            { get
                { return Properties.Resources.MainWindow_LanguageCzech; }
            }
        public string LanguageEnglish
            { get
                { return Properties.Resources.MainWindow_LanguageEnglish; }
            }
        public string ExitApp
            { get { return Properties.Resources.MainWindow_Exit; } }
        public string AddContent
            { get { return Properties.Resources.MainWindow_AddContent; } }
        public string AddPicture
```

```

    { get { return Properties.Resources.MainWindow_AddPicture; } }
public string AddPresentation
    { get
      { return Properties.Resources.MainWindow_AddPresentation; }
    }
public string AddExcelGraph
    { get
      { return Properties.Resources.MainWindow_AddExcelGraph; }
    }
public string AddExcelTable
    { get
      { return Properties.Resources.MainWindow_AddExcelTable; }
    }
public string AddWwwReference
    { get
      { return Properties.Resources.MainWindow_AddWwwReference; }
    }
public string AddVideo
    { get { return Properties.Resources.MainWindow_AddVideo; } }
public string AddVisitor
    { get { return Properties.Resources.MainWindow_AddVisitor; } }
#endregion
#region Commands
public ICommand NewVisuObjectCommand
{
    get
    {
        return new RelayCommand(param
            => this.CreateNewVisuObject());
    }
}
RelayCommand _visuPicturesCommand;
public ICommand VisuPicturesCommand
{
    get
    {
        if (_visuPicturesCommand == null)
        {
            _visuPicturesCommand = new RelayCommand(param
                => this.ShowAllVisuPictures());
        }
        return _visuPicturesCommand;
    }
}
public ICommand SaveFilesInActiveWorkspaceToDb
{
    get
    {
        return new RelayCommand(param
            => this.SaveFilesInActiveWorkspace());
    }
}
RelayCommand _lcdLayoutsCommandAdmin;
public ICommand LcdlayoutsCommandAdmin
{
    get
    {
        if (_lcdLayoutsCommandAdmin == null)
        {
            _lcdLayoutsCommandAdmin = new RelayCommand(param
                => this.ShowAllLcdLayouts(true));
        }
    }
}

```

```

        return _lcdLayoutsCommandAdmin;
    }
}
RelayCommand _lcdLayoutsCommand;
public ICommand LcdlayoutsCommand
{
    get
    {
        if (_lcdLayoutsCommand == null)
        {
            _lcdLayoutsCommand = new RelayCommand(param
                => this.ShowAllLcdLayouts(false));
        }
        return _lcdLayoutsCommand;
    }
}
RelayCommand _lcdLayoutsCyclesCommand;
public ICommand LcdLayoutsCyclesCommand
{
    get
    {
        if (_lcdLayoutsCyclesCommand == null)
        {
            _lcdLayoutsCyclesCommand = new RelayCommand(param
                => this.ShowLcdCyclesLayouts());
        }
        return _lcdLayoutsCyclesCommand;
    }
}
}
#endregion
#region Private Helpers
void ShowAllVisuPictures()
{
    VisuPicturesViewModel workspace
        = _allWorkspaces.Workspaces.FirstOrDefault(vm
            => vm is VisuPicturesViewModel) as VisuPicturesViewModel;
    if (workspace == null)
    {
        workspace = new VisuPicturesViewModel(_allWorkspaces,
            _userId);
        _allWorkspaces.Workspaces.Add(workspace);
    }
    _allWorkspaces.SetActiveWorkspace(workspace);
}
void CreateNewVisuObject()
{
    switch (ActiveWorkspaceName())
    {
        case "VisuPicturesViewModel":
            VisuPictureViewModel workspace
                = _allWorkspaces.Workspaces.FirstOrDefault(vm
                    => vm is VisuPictureViewModel)
                as VisuPictureViewModel;
            VisuPicture newVisuPicture
                = VisuPicture.CreateNewVisuPicture();
            VisuPictureRepository visuPictureRepository
                = (VisuPictureRepository)
                GetActiveWorkspaceRepository();
            workspace = new VisuPictureViewModel(_allWorkspaces,
                newVisuPicture, visuPictureRepository);
            _allWorkspaces.Workspaces.Add(workspace);
            _allWorkspaces.SetActiveWorkspace(workspace);
            break;
    }
}

```

```
    }  
  }  
  void SaveFilesInActiveWorkspace()  
  {  
    switch (ActiveWorkspaceName())  
    {  
      case "VisuPicturesViewModel":  
        break;  
    }  
  }  
#endregion  
}  
}
```

PŘÍLOHA P XI. MAINWINDOWWORKSPACESVIEWMODEL.CS

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;
namespace ControlInterfaceWopvs.ViewModels
{
    public class MainWindowWorkspacesViewModel : WorkspaceViewModel
    {
        #region Fields
        ObservableCollection<WorkspaceViewModel> _workspaces;
        #endregion
        #region Public methods
        public void SetActiveWorkspace(WorkspaceViewModel workspace)
        {
            Debug.Assert(_workspaces.Contains(workspace));
            ICollectionView collectionView
                = CollectionViewSource.GetDefaultView(_workspaces);
            if (collectionView != null)
                collectionView.MoveCurrentTo(workspace);
        }
        #endregion
        #region Workspaces
        /// <summary>
        /// Returns the collection of available workspaces to display.
        /// A 'workspace' is a ViewModel that can request to be closed.
        /// </summary>
        public ObservableCollection<WorkspaceViewModel> Workspaces
        {
            get
            {
                if (_workspaces == null)
                {
                    _workspaces
                        = new ObservableCollection<WorkspaceViewModel>();
                    _workspaces.CollectionChanged
                        += this.OnWorkspacesChanged;
                }
                return _workspaces;
            }
        }
        void OnWorkspacesChanged(object sender,
            NotifyCollectionChangedEventArgs e)
        {
            if (e.NewItems != null && e.NewItems.Count != 0)
                foreach (WorkspaceViewModel workspace in e.NewItems)
                    workspace.RequestClose
                        += this.OnWorkspaceRequestClose;
            if (e.OldItems != null && e.OldItems.Count != 0)
                foreach (WorkspaceViewModel workspace in e.OldItems)
                    workspace.RequestClose
                        -= this.OnWorkspaceRequestClose;
        }
        void OnWorkspaceRequestClose(object sender, EventArgs e)
        {
            WorkspaceViewModel workspace = sender as WorkspaceViewModel;
```

```
        workspace.Dispose();
        this.Workspaces.Remove(workspace);
    }
#endregion
}
}
```

PŘÍLOHA P XII. VISUBASEOBJECT.CS

```
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Text.RegularExpressions;
namespace ControlInterfaceWopvs.Models
{
    public class VisuBaseObject : IDataErrorInfo
    {
        #region Creation
        #endregion // Creation
        #region State properties
        /// <summary>
        /// Visualisation object Id in DB - if Id == 0
        /// then it is not saved in DB yet
        /// </summary>
        public int Id { get; set; }
        /// <summary>
        /// Name of the visualisation object
        /// </summary>
        public string FileName { get; set; }
        /// <summary>
        /// Full path of the visualisation object; it is important
        /// for automatic update (It has to be public network space)
        /// </summary>
        public string FilePath { get; set; }
        /// <summary>
        /// Binary content of the visualisation object
        /// </summary>
        public byte[] FileContent { get; private set; }
        /// <summary>
        /// Date and time of visualisation object creation
        /// </summary>
        public DateTime CreatedAt { get; set; }
        /// <summary>
        /// Name of person who saved the visualisation object to DB
        /// for the first time
        /// </summary>
        public string AuthorName { get; set; }
        /// <summary>
        /// Name of person who saved updated visualisation object
        /// to DB for the last time
        /// </summary>
        public string LastEditorName { get; set; }
        /// <summary>
        /// Logical value defines if the visualisation
        /// object will be automatically updated (It has to be
        /// in public network space)
        /// </summary>
        public bool AutoUpdate { get; set; }
        #endregion // State properties
        #region IDataErrorInfo Members
        string IDataErrorInfo.Error
        {
            get { return null; }
        }
        string IDataErrorInfo.this[string propertyName]
        {
```

```

        get { return this.GetValidationError(propertyName); }
    }
#endregion // IDataErrorInfo Members
#region Validation
/// <summary>
/// Returns true if this object has no validation errors.
/// </summary>
public bool IsValid
{
    get
    {
        foreach (string property in ValidatedProperties)
            if (GetValidationError(property) != null)
                return false;
        return true;
    }
}
static readonly string[] ValidatedProperties =
{
    "FileName",
    "FilePath",
    "CreatedAt"
};
string GetValidationError(string propertyName)
{
    if (Array.IndexOf(ValidatedProperties, propertyName) < 0)
        return null;
    string error = null;
    switch (propertyName)
    {
        case "FileName":
            error = this.ValidateFileName();
            break;
        case "FilePath":
            error = this.ValidateFilePath();
            break;
        case "CreatedAt":
            error = this.ValidateCreatedAt();
            break;
        default:
            Debug.Fail("Unexpected property being validated on Customer: "
                + propertyName);
            break;
    }
    return error;
}
string ValidateFileName()
{
    if (IsStringMissing(this.FileName))
    {
        return
            LocalResources.ModelsVisu.ValidationErrorMessage_MissingFileName;
    }
    return null;
}
string ValidateFilePath()
{
    if (IsStringMissing(this.FilePath))
    {
        return
            LocalResources.ModelsVisu.ValidationErrorMessage_MissingFilePath;
    }
    else if (!IsValidPathOfFile(this.FilePath))
    {
        return
            LocalResources.ModelsVisu.ValidationErrorMessage_InvalidFilePath;
    }
}

```

```

        }
        return null;
    }
    string ValidateCreatedAt()
    {
        if (IsDateTimeMissing(this.CreatedAt))
        {
            return
                LocalResources.ModelsVisu.ValidationErrorMessage_MissingCreationDT;
        }
        return null;
    }
    static bool IsStringMissing(string value)
    {
        return
            String.IsNullOrEmpty(value) ||
            value.Trim() == String.Empty;
    }
    static bool IsDateTimeMissing(DateTime value)
    {
        return value == null;
    }
    static bool IsValidPathOfFile(string pathOfFile)
    {
        if (IsStringMissing(pathOfFile))
            return false;
        string pattern = @"...";
        return Regex.IsMatch(pathOfFile, pattern,
            RegexOptions.IgnoreCase);
    }
    #endregion // Validation
    #region Public methods
    public void ReadFileByteContent()
    {
        byte[] dataOfFile;
        DateTime dateTimeOfFile;
        string nameOfFile;
        if (!StandardTools.FilesRW.ReadFile(FilePath, out dataOfFile,
            out dateTimeOfFile, out nameOfFile))
            throw new Exception(
                LocalResources.ModelsVisu.FileContentReadingError + " "
                + this.FileName);
        this.FileContent = dataOfFile;
        this.CreatedAt = dateTimeOfFile;
    }
    #endregion // Public methods
}
}
}

```

PŘÍLOHA P XIII. LCD_S_MAINCYCLE

```
USE [WOPVS_lcd_app1]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[LCD_s_MainCycle]
@LcdLayoutId int
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
SET NOCOUNT ON;
    -- Insert statements for procedure here
SELECT * FROM dbo.v_LCD_MainAreaCycles_Limitation
WHERE LcdLayoutId = @LcdLayoutId
AND (
(DateTimeLimitationId IS NULL)
OR (
((CycleDaysId IS NULL) OR (CycleDays like '
AND ((CycleDatesId IS NULL)
        OR ((CycleStartDate < CAST(GETDATE() AS date))
            AND (CycleStopDate > CAST(GETDATE() AS date))))
AND ((CycleTimesId IS NULL)
        OR ((CycleStartTime < CAST(GETDATE() AS time))
            AND (CycleStopTime > CAST(GETDATE() AS time))))
)
)
END
```

PŘÍLOHA P XIV. CALENDAR_DAYOFWEEK

```
USE [WOPVS_1cd_app1]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER FUNCTION [dbo].[Calendar_DayOfWeek]
(
    @dtDate DateTime          -- Current date
)
RETURNS nvarchar(2)
AS
BEGIN
    -- Variables
    declare @intDayOfWeek    int;
    -- Get the day of week
    set @intDayOfWeek = (((@@datefirst-1)+datepart(weekday,@dtDate)) % 7);
    -- Calculate the offset
    return CASE @intDayOfWeek
    WHEN 0 THEN 'Ne'
    WHEN 1 THEN 'Po'
    WHEN 2 THEN 'Ut'
    WHEN 3 THEN 'St'
    WHEN 4 THEN 'Ct'
    WHEN 5 THEN 'Pa'
    WHEN 6 THEN 'So'
    ELSE 'Er'
    END
END
```

PŘÍLOHA P XV. COMPUTERAREACONTAINERVEIWMODEL

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VisualisationTools.DataAccess;
using VisualisationTools.Model;
namespace VisualisationTools.ViewModel
{
    public class ComputerAreaContainerVeiwModel : WorkspaceViewModel
    {
        #region Fields
        readonly LcdLayoutsVisualisationRepository _lcdLayoutsRepository;
        ObservableCollection<WorkspaceViewModel> _lcdPanelsWorkspaces;
        readonly string _computerName;
        readonly int _visualisationRatio;
        readonly int _userId;
        #endregion
        #region Constructor
        public ComputerAreaContainerVeiwModel(string computerName,
            int visualisationRatio, int userId)
        {
            _computerName = computerName;
            _visualisationRatio = visualisationRatio;
            _userId = userId;
            _lcdLayoutsRepository
                = new LcdLayoutsVisualisationRepository(_computerName,
                    _userId);
            LoadLcd();
        }
        #endregion
        #region Presentation properties
        WorkspaceViewModel lcd1;
        public WorkspaceViewModel Lcd1 { get { return lcd1; } }
        #endregion
        #region Private helpers
        void LoadLcd()
        {
            foreach (LayoutContainer lcd
                in _lcdLayoutsRepository.GetLayoutContainers())
            {
                if (lcd.Id == 2) // test
                    lcd1 = new LcdAreaContainerViewModel(lcd,
                        _visualisationRatio, _userId);
            }
        }
        #endregion
    }
}
```

PŘÍLOHA P XVI. LCDAREACONTAINERVIEWMODEL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VisualisationTools.DataAccess;
using VisualisationTools.Model;
namespace VisualisationTools.ViewModel
{
    public class LcdAreaContainerViewModel : WorkspaceViewModel
    {
        #region Fields
        readonly LayoutContainer _thisLayoutSetup;
        readonly int _visualisationRatio;
        readonly int _userId;
        WorkspaceViewModel _mainAreaWorkspace;
        #endregion
        #region Constructor
        public LcdAreaContainerViewModel(LayoutContainer thisLayoutSetup,
            int visualisationRatio, int userId)
        {
            _thisLayoutSetup = thisLayoutSetup;
            _visualisationRatio = visualisationRatio;
            _userId = userId;
            LoadMainArea();
        }
        #endregion
        #region Presentation properties
        public WorkspaceViewModel MainAreaWorkspace
        { get { return _mainAreaWorkspace; } }
        #endregion
        #region Private helpers
        void LoadMainArea()
        {
            _mainAreaWorkspace
                = new MainAreaContainerViewModel(_thisLayoutSetup.Id,
                    _userId);
        }
        #endregion
    }
}
```

PŘÍLOHA P XVII. MAINAREACONTAINERVIEWMODEL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VisualisationTools.DataAccess;
using VisualisationTools.Model;
namespace VisualisationTools.ViewModel
{
    public class MainAreaContainerViewModel : WorkspaceViewModel
    {
        #region Fields
        readonly int _lcdLayoutId;
        readonly int _userId;
        readonly MainAdditionalAreaContainerRepository _mainAreaRepository;
        WorkspaceViewModel _mainAreaWorkspace;
        #endregion
        #region Constructor
        public MainAreaContainerViewModel(int lcdLayoutId, int userId)
        {
            _lcdLayoutId = lcdLayoutId;
            _userId = userId;
            _mainAreaRepository
                = new MainAdditionalAreaContainerRepository(_lcdLayoutId,
                    LcdVisualisation.LcdAreaElementVisualisation.MainArea,
                    _userId);
            System.Threading.Thread t = new System.Threading.Thread(Play);
            t.Start();
        }
        #endregion
        #region Presentation properties
        public WorkspaceViewModel MainAreaWorkspace
        { get { return _mainAreaWorkspace; } }
        #endregion
        #region Private helpers
        private void Play()
        {
            foreach (MainAdditionalAreaContainer container
                in _mainAreaRepository.GetMainAdditionalAreaContainers())
            {
                if (container.PictureId != 0)
                {
                    _mainAreaWorkspace
                        = new PictureViewModel(container.PictureId);
                    base.OnPropertyChanged("MainAreaWorkspace");
                }
                System.Threading.Thread.Sleep(container.VisibleSeconds * 1000);
            }
        }
        #endregion
    }
}
```