VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Soubor zásuvných modulů pro systém ForceB - Webinar

Collection of Plug-ins for ForceB - Webinar System

2016 Petr Rojko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Rojko**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Soubor zásuvných modulů pro systém ForceB - Webinar
Collection of Plug-ins for ForceB - Webinar System

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem bakalářské práce je navrhnout a implementovat nové zásuvné moduly pro systém ForceB – Webinar, které rozšíří jeho funkcionalitu. Bude se jednat o celkem tři zásuvné moduly umožňující tutorovi v rámci webového semináře sdílet PDF soubory, prezentace pptx vytvořené pomoci Microsoft PowerPoint a dále se bude jednat o modul zprostředkovávající interaktivní tabuli tzv. whiteboard. Zároveň bude cílem bakalářské práce optimalizovat současné zásuvné moduly pro sdílení pracovní plochy a sdílení okna aplikace.

Během řešení postupujte podle následujících bodů:
1. Popište systém pro pořádání webových seminářů ForceB – Webinar.
2. Prostudujte a popište současné zásuvné moduly. Zaměřte se na rozhraní umožňující přidávání modulů formou pluginu. Navrhněte a implementujte úpravy modulů, které povedou k optimalizaci při získávání a zpracování obrazu.
3. Vytvořte analýzu a návrh jednotlivých zásuvných modulů, které rozšíří funkčnost stávajícího systému.
4. Jednotlivé moduly budou naimplementovány na platformě .NET Framework formou dll knihoven.
5. Proveďte ověření výsledných modulů v reálném provozu a zhodnoťte výsledky.

Seznam doporučené odborné literatury:

[1] PROKEŠ, Martin a Radoslav FASUGA. ICETA 2012 proceedings 10th IEEE International Conference on Emerging eLearning Technologies and Applications: November 8-9, 2012, Stará Lesná, The High Tatras, Slovakia. Piscataway, N.J: IEEE, 2012, Tool for desktop sharing and remote teaching - For. ISBN 9781467351201.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Prokeš**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Rád bych poděkoval Ing. Martinu Prokešovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Následující bakalářská práce se zabývá návrhem a realizací nových zásuvných modulů pro systém ForceB, umožňujících tutorovi v rámci webového semináře prezentovat PDF a PowerPoint prezentace. Dále přináší whiteboard modul neboli interaktivní tabuli pro kreslení. Také se zabývá optimalizací a úpravou stávajících zásuvných modulů, konkrétně modulů sdílení plochy a okna aplikace a to vzhledem k efektivitě získávání obrazu a jeho zpracování. Součástí práce je i popis systému ForceB spolu s rozhraním pro zásuvné moduly. Veškerá funkcionalita je postavena na .NET Frameworku od firmy Microsoft.

Klíčová slova: Webinář, E-learning, Zásuvný modul, Microsoft PowerPoint, PDF, DLL, .NET Framework, Interaktivní tabule, DirectX, GDI+

Abstract

The following work deals with design and implementation of new plugins for system ForceB, allowing the tutor within the webinar to present PDF and PowerPoint presentations. Also brings a whiteboard plug-in or interactive whiteboard for drawing functionality. It also deals with optimization of existing plug-ins, namely the desktop and application window sharing due to efficiency of image acquisition and image processing. The work also includes a description of the ForceB system along with an interface description for plug-ins.Whole functionality is built on top of the .NET Framework from Microsoft.

Key Words: Webinar, E-learning, Plugin, Microsoft PowerPoint, PDF, DLL, .NET Framework, Whiteboard, DirectX, GDI+

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
C++/CLI	–	Common Language Infrastructure - Microsoft C++ jazyk
CPU	–	Central Processing Unit
CUDA	–	Compute Unified Device Architecture
DLL	–	Dynamic Link Library
FPS	–	frames per second - snímky za sekundu
FullHD	–	Full High-Definition - rozlišení 1920x1080 pixelů
GDI/GDI+	–	Graphics Device Interface
GPU	–	Graphic Processing Unit
GUI	–	Grafické uživatelské rozhraní
HD	–	High-Definition - rozlišení 1280x720 pixelů
HW	–	Hardware
ID	–	Identifikátor
IP	–	Internet Protocol
IT	–	Informační technologie
Mb	–	Megabit
NET Framework	–	"dotnet" - softwarová platforma
OS	–	Operation System
PDF	–	Portable Document Format
PPT/PPTX	–	Microsoft Office Power Point
RAM	–	Random Access Memory
SHA1	–	Secure Hash Algorithm 1 - hašovací funkce
UI	–	Uživatelské rozhraní
VOIP	–	Voice over IP
kb	–	kilobit

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Vývoj v oblasti informačních technologií oproti předchozím letům znovu značně pokročil. Dnes a denně můžeme pozorovat stále nové rozvíjející se technologie. Hardware je stále výkonnější a rychlejší, propustnost přenosových sítí se každým rokem zvyšuje. Celkový pokrok v těchto oblastech spěje postupně k tomu, že se tyto věci stávají stále dostupnější širší množině koncových uživatelů, kteří v rámci svých možností mohou vlastnit stále lepší technologie. Dnešní uživatelé i vývojáři softwaru již nejsou tedy tak limitováni či omezeni dřívějšími technologiemi. Stejně tak pokrok v oblasti přístupových sítí vedl ke zvýšení průměrné rychlosti nabízené poskytovateli připojení k internetu. A tento trend každým rokem pokračuje, zvláště v Evropě a USA. Je tedy možné vyvíjet a provozovat aplikace či služby s mnohem většími požadavky na výkon hardwaru a nároky na datové přenosy, než bylo dříve zvykem. Stále častější a oblíbenější jsou například streamované videopřenosy v reálném čase. Podobné služby jsou mezi uživateli internetu stále žádanější. Zmíněný rozvoj a poptávka po takovýchto službách nás nutí vyvíjet multimediální aplikace nabízející širokou škálu funkcionality, jako jsou webové semináře nebo videokonference.

Přenos multimediálních dat pomocí zásuvných modulů, zpracování obrazu a téma webových seminářů, to vše je předmětem mé bakalářské práce. Touto prací navazuji na již vyvinutý systém ForceB soustředící se na pořádání webových seminářů. Hlavním cílem práce je rozvinout tento systém dále pomocí nové funkcionality přidané zásuvnými moduly. Vytvoření nového rozhraní pro zavádění a tvorbu modulů. Přepis původních modulů pracujících s obrazem. Optimalizace procesu zpracování obrazu. A následná analýza a srovnání původních modulů s novými, spolu s provedenými optimalizacemi. Motivací bylo systém vylepšit o nové prvky funkcionality a optimalizovat jej a jeho stávající funkce do užitečnější a obšírněji použitelné podoby.

Na začátku se práce věnuje krátkému úvodu do webových seminářů a e-learningu, kde popíše výhody a jejich možnosti. Dále následuje část popisující některé současné systémy nabízející nebo mající funkcionalitu webových seminářů. Následuje stručný popis systému ForceB pro pořádání webových seminářů a jeho desktopového klienta. U klienta se jedná hlavně o popis jeho grafického rozhraní a stručné srovnání s novou odlehčenou verzí. Další kapitolou v pořadí jsou původní zásuvné moduly pracující s obrazem. Zde je popis původních modulů, jejich funkcionality a grafického rozhraní. Následuje porovnání původních modulů oproti nově implementovaným a optimalizovaným modulům a analýza jejich výkonu. Text se dále přesunuje k novým modulům a nově navrženému rozhraní pro jejich zavedení. U rozhraní zmiňuje jeho přínos a způsob jakým jej lze použít pro vývoj nových modulů. Dalším bodem jsou nové zásuvné moduly - jejich popis, funkcionalita a analýza. Druhá část práce se zaměřuje na optimalizace a samotný proces zpracování obrazu. Zde se snaží srovnat dřívější řešení s novým, popsat celý proces, jednotlivé provedené optimalizace a přiblížit jednotlivé komponenty a jejich třídy a fungování. Poslední část práce se zaobírá popisem, funkcionalitou a optimalizacemi jednotlivých mnou vytvořených knihoven použitých v jednotlivých zásuvných modulech.

2 E-learning a systém ForceB

V této kapitole rozvedu téma webových seminářů a e-learningu. Představím existující řešení na trhu spolu se systémem ForceB pro pořádání webových seminářů. Nakonec představím předělané grafické rozhraní systému ForceB s jeho možnostmi.

2.1 Webový seminář a e-learning

Slovo webový seminář je odvozené ze slova Web-based seminar. Může to být prezentace, lekce, seminář, přednáška nebo workshop, který je přenášený skrze internet. Seminář je veden vždy nějakým lektorem. Komunikace bývá obvykle oboustranná a přenos zajišťuje videokonferenční software. Hlavní výhodou je právě interaktivita mezi uživateli, je možné diskutovat a vyměňovat si informace v reálném čase. To nám dává možnost realizovat formu elektronické výuky a sdílet informace a poznatky mezi ostatními uživateli. K provozu stačí běžný hardware a konkrétní software realizující přenos. Obvykle mohou uživatelé sdílet webovou kameru, soubory, hlas nebo aktuální pracovní plochu. Některé systémy nabízejí možnost pořádat během semináře ankety nebo uložit záznam semináře do souboru. Záznam lze poté později sdílet s konkrétním okruhem lidí nebo jej publikovat veřejně. Stejně tak některé systémy nabízejí virtuální místnosti pro uživatele konkrétního kurzu a další vymoženosti. Celkově přináší webové semináře a e-learning velkou úsporu času, není třeba nikam jezdit nebo chodit. To přináší úsporu nákladů na dopravu. Taktéž se mohou setkat lidé, kteří by se za normálních okolností setkat nemohli. Nespornou výhodou pro pořadatele kurzu je, že nemusejí pronajímat fyzickou místnost, stačí jim virtuální. Podobným výčtem výhod bychom mohli pokračovat dále. Jedinou nevýhodou, kterou naopak takové systémy přinášejí, je, že se vše odehrává pouze virtuálně.

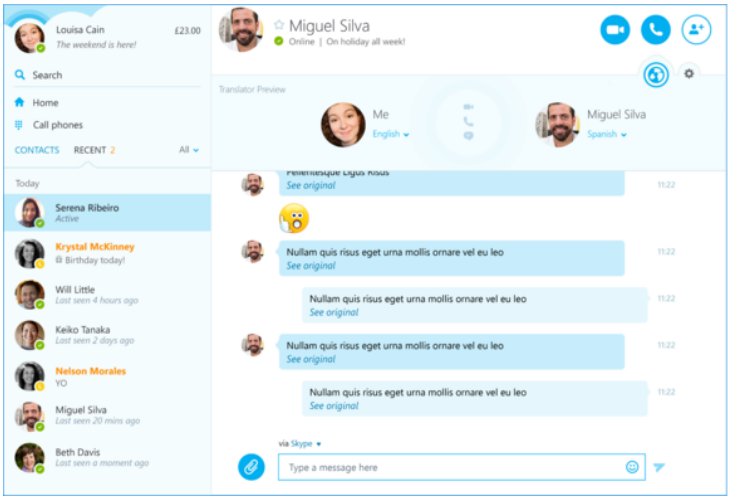
Od doby vzniku webových seminářů, vzniklo hned několik platforem, které se na danou problematiku více či méně zaměřují. Jednou z těchto platforem je systém ForceB, který se přímo zaměřuje na tuto funkcionalitu, a který zmíním níže po stručném představení existujících řešení na trhu.

2.2 Existující řešení na trhu

Na trhu již v dnešní době existuje docela nemalé množství řešení, která nabízejí nějakou formu videokonference či e-learningu. Tyto pojmy jsou si poměrně blízké, jelikož oba svým způsobem nabízejí obdobné funkce, vždy jde o použití web kamery, sdílení plochy a přenos hlasu. Občas se k tomu všemu ještě přidává podpora ve formě chatu a odesílání souborů, případně další funkcionalita. Protože jsou tyto systémy již popsány v předešlé práci mého vedoucího a nedošlo k žádné výrazné změně na jejich poli, tak je jen stručně popíši a zmíním jejich rozdíly. Případné zájemce o jejich detailnější popis odkážu na předchozí bakalářskou nebo diplomovou práci mého vedoucího.

2.2.1 Skype

Dnes asi jeden z nejznámějších a nejpoužívanějších programů pro komunikaci mezi uživateli. Často užívaný k výuce na dálku, třeba jako nástroj pro doučování matematiky. Jeho nespornou výhodou je jeho dostupnost, kterou udává to, že je nabízen zdarma a vlastní jej většina uživatelů PC. Navíc nabízí prakticky všechny možnosti přenosu, umí sdílet plochu, hlas, web kameru, nabízí chat, odesílání souborů a další vymoženosti. Jeho grafické rozhraní působí navíc docela intuitivně, viz níže.



Obrázek 1: UI rozhraní Skypu

Shrnutí možností

- Poměrně přehledné a intuitivní UI rozhraní.
- VOIP - přenos hlasu a pořádání hromadných konferencí.
- Přenos web kamery.
- Sdílení plochy uživatele.
- Přenos souborů.
- Chat rozhraní.
- Dostupný zdarma.

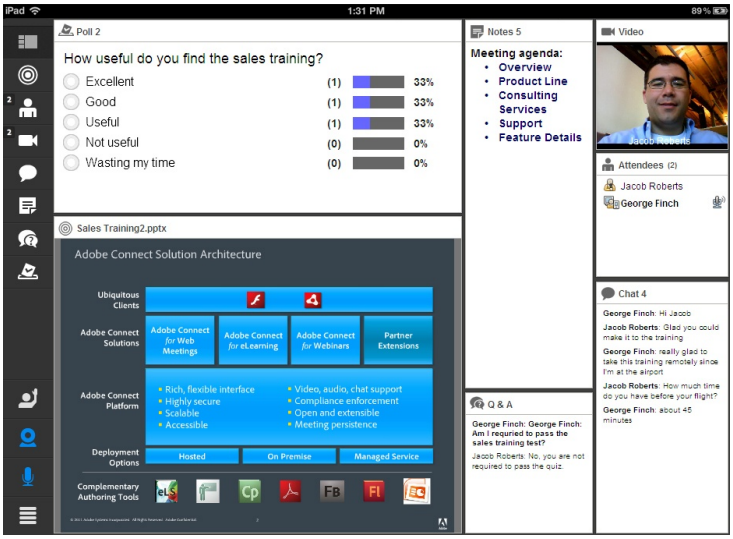
Skype tak nabízí prakticky všechny druhy dnes užívaných datových přenosů. Jediné, co zde chybí, je bližší specializace na e-learning a webové semináře. Dále také specializace v podobě virtuálních místností, učeben, kurzů a přídavných modulů věnujících se této funkcionalitě.

2.2.2 Adobe Acrobat Connect

Je software určený pro vytváření informací a prezentací, online tréninkových materiálů, web konferencí, sdílení plochy uživatele a učebních zásuvných modulů. Produkt je založen kompletně nad Adobe Flash platformou, to mu umožňuje běžet multiplatformně. Na poli podobně založených produktů je pravděpodobně hlavním lídrem určujícím směr. To dokazuje Adobe i na svých stránkách na základě grafů ohledně nejlepšího webinářového systému. Jde o hodně rozsáhlé a robustní řešení určené pro velké a střední firmy. Práce probíhá pomocí konferenčních místností. Systém se skládá z následujících aplikací.

- **Adobe Connect Meetings**
- **Adobe Connect Webinars**
- **Adobe Connect Learning**

Systém dále nabízí prakticky všechny formy sdílení a přenosu dat. Jeho grafické rozhraní prošlo četným vývojem, viz níže.



Obrázek 2: UI rozhraní Adobe Connect

Shrnutí možností

- Prakticky všechny možnosti přenosů dat a jejich sdílení.
- Složitější UI rozhraní s množstvím výhod.
- Zabezpečený přenos dat.
- Přístup skrze různá zařízení.
- Zabudovaná podpora Analytics.

- Dostupný více pro firmy, vysoké poplatky, není zdarma .

Kompletní řešení, kterému prakticky nechybí nic k dokonalosti. Vysoce nastavitelné s podporou mnoha dodatečných nástrojů a rozšíření.

2.2.3 Cisco WebEx

Dle velikosti řešení jeho robustnosti a hodnocení na poli podobných systému je Cisco WebEx druhé v pořadí. Firma Cisco nabízí vícero řešení založených pod značkou Cisco WebEx, těmi jsou:

- **WebEx Meetings** - nabízeno v bezplatné a prémiové verzi pro organizaci mítinků.
- **WebEx Event Center** - tvorba událostí a webových seminářů.
- **WebEx Training Center** - pořádání online kurzů a tréninků.
- **WebEx Support Center** - pro nabídku online vzdálené podpory.

Cisco WebEx se chlubí podobnými vymoženostmi jako předchozí představené řešení na trhu. Umožňuje sdílet plochu, využívat VOIP služby, whiteboard a další. Na rozdíl od Adobe Connect je možné jej provozovat omezeně i zdarma a lze si vyzkoušet i plnou 14 denní verzi. Software je určen středním a velkým firmám, ale nezvládne takové množství uživatelů jako řešení od Adobe. Jeho grafické rozhraní doznalo taktéž mnoha změn.

Obrázek 3: UI rozhraní Cisco WebEx

Shrnutí možností

- Možno vyzkoušet 14 denní plnou verzi, lze provozovat omezeně i zdarma.
- Zabezpečený přenos dat.

- Nižší poplatky za provoz.
- Srovnatelné možnosti přenosu a sdílení dat.

2.3 Popis systému ForceB

Systém navržený pro videokonference a pořádání webových seminářů. Skládá se z komunikačního serveru, případných komunikačních uzlů zajištujících rozložení výkonu na síti a grafického klienta na straně každého uživatele. Server i grafický klient využívají sadu knihoven pro komunikaci skrze proprietární komunikační protokol stejného názvu jako systém.

2.3.1 Desktopový klient

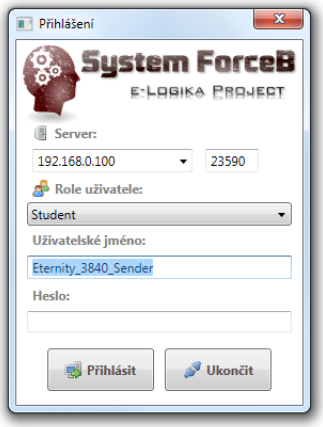
Grafický klient zajištující funkcionalitu na straně každého uživatele systému ForceB. Je tvořen sadou zásuvných modulů a knihoven zajištujících jeho funkcionalitu a případnou funkcionalitu modulů. Klient zajišťuje funkce pro přenos webové kamery, VOIP nebo zpráv z chatovacího rozhraní. Druhou část funkcionality klienta zajišťují jednotlivé zásuvné moduly. Klient je tímto plně rozšiřitelný a koncovým uživatelům tak stačí pouze vlastnit konkrétní modul umístěný ve složce aplikace, odtud totiž aplikace jednotlivé moduly zavádí.

Klient samotný operuje ve dvou módech a to v módu studenta a učitele.

- **Student** - přijímá data od učitele a zobrazuje je (prezentace, sdílení plochy), reprezentuje roli posluchače v učebně.
- **Učitel** - odesílá a streamuje data do sítě studentům. Representuje roli tutora a přednášejícího.

Vybraný mód je určen uživatelem při přihlášení do aplikace, kdy uživatel vybere, zda se přihlašuje jako student nebo učitel, podle toho se uzpůsobí i UI aplikace. V módu studenta aplikace skryje nabídku pro spuštění jednotlivých zásuvných modulů a zobrazí uprostřed okna aplikace box určený pro UI modulů. Moduly nabízející UI tak budou zobrazeny v tomto boxu. V módu učitele naopak aplikace skryje box a jeho místo připadne boxu s chat funkcionalitou. Je třeba dodat, že některé moduly jako kreslící tabule, zobrazí své UI rozhraní i v módu učitele, finální slovo má tak vždy konkrétní modul.

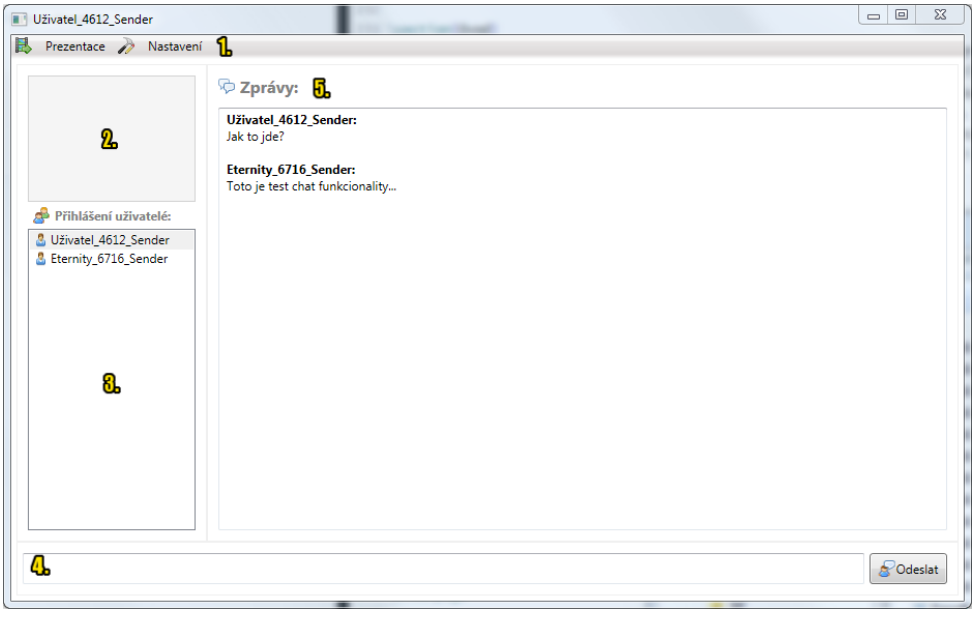
Klient při zavádění modulu sdělí modulu mód uživatele, ve kterém se aktuálně nachází. Modul na základě toho přizpůsobí své chování. Samozřejmě není nikde striktně určeno, že by modul v módu studenta nemohl odesílat data a v módu učitele je naopak přijímat. Každý modul si tímto způsobem odděluje nebo uzpůsobuje funkcionalitu pro vybraný mód, ve kterém běží pod uživatelem. Samotný přihlašovací dialog s volbou módu můžeme vidět níže na obrázku 4.



Obrázek 4: Přihlašovací dialog pro aplikaci

V dialogu můžeme vidět IP adresu serveru, ke které se chceme připojit, použitý port, výběr výše zmíněné role uživatele, přihlašovací jméno a heslo. Použité adresy serverů jsou navíc pamatovány pro příště.

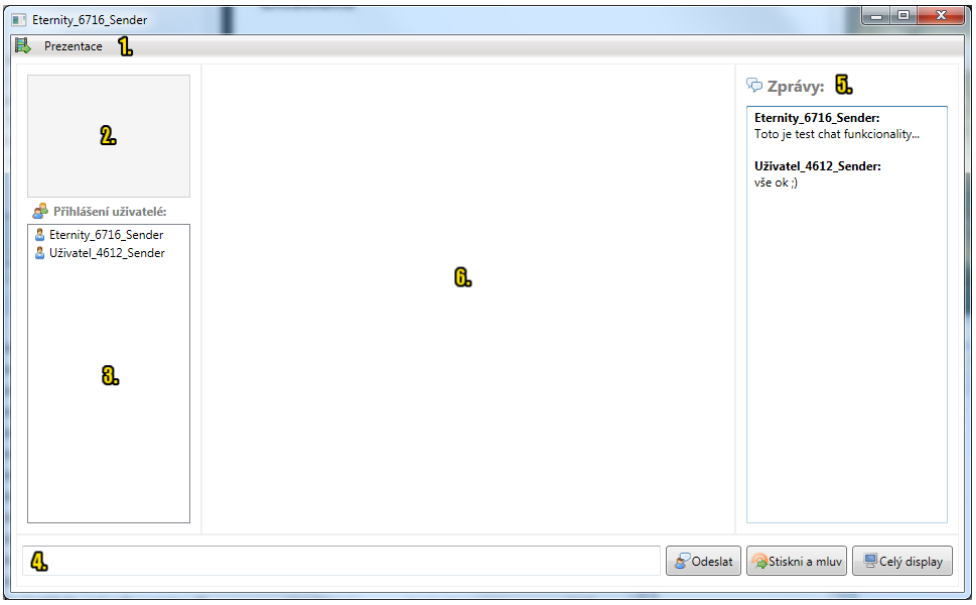
Po úspěšném přihlášení do systému ForceB se nám zobrazí okno klienta připravené plnit naše požadavky. Před jeho popisem je třeba zmínit, že bylo grafické rozhraní částečně přepracováno a kód aplikace kompletně refaktorován do nové podoby. Většina původního kódu tak byla odstraněna a nahrazena. Původní aplikace totiž nebyla pro nové zásuvné moduly a přepracování stávajících vhodná z toho důvodu, že její kód působil částečně nepřehledně a byl nekompatibilní. Grafické rozhraní i kód tedy doznali výrazného odlehčení do dnešní podoby.



Obrázek 5: Klient v módu učitele.

Popis funkcionality v módu učitele

1. Vrchní menu aplikace - obsahuje nabídku pro pořádání webových prezentací a nabídku s nastavením aplikace.
2. Okno pro web kameru - pokud je aktivována web kamera, zobrazí se právě v tomto okně.
3. Seznam aktuálně připojených uživatelů - jednotliví uživatelé v kurzu.
4. Spodní lišta - obsahuje box spolu s tlačítkem pro odesílání zpráv.
5. Okno pro chat komunikaci - zobrazuje zprávy uživatelů.

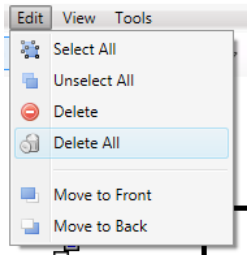


Obrázek 6: Klient v módu studenta.

Popis funkcionality v módu student

1. Vrchní menu aplikace - pouze menu prezentací, student nemůže zasahovat do nastavení.
2. Okno pro web kameru - pokud je aktivována web kamera, zobrazí se právě v tomto okně.
3. Seznam aktuálně připojených uživatelů - jednotliví uživatelé v kurzu.
4. Spodní lišta - obsahuje box pro chat spolu s tlačítkem pro odeslání, tlačítko pro VOIP přenos a tlačítko pro mód celé obrazovky, když běží prezentace.
5. Okno pro chat komunikaci - zobrazuje zprávy uživatelů.
6. Okno pro prezentaci - zde daný modul zobrazí své UI rozhraní pro studenta.

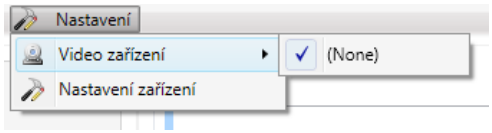
Dále zde máme nabídku pod názvem prezentace, která umožňuje ovládat zvuk, web kameru a jednotlivé zásuvné moduly.



Obrázek 7: Nabídka prezentace

- **Spustit prezentaci** - zobrazí nabídku zásuvných modulů pro vyučujícího, zobrazeny jsou pouze nalezené moduly, uživatel tak může kdykoli přidat nové.
- **Zastavit prezentaci** - zastavuje aktuální prezentaci, prezentace je odstraněna z UI.
- **Použít web kameru** - přepínač, který umožňuje zapnout odběr web kamery.
- **Použít zvuk** - přepínač, který umožňuje zapnout odběr zvuku (VOIP).
- **Ukončit** - ukončení aplikace a prezentace.

Další nabídkou je nastavení, to umožňuje zvolit zařízení webové kamery a jeho případné alternativy nabízené OS. Také zde najdeme nastavení aplikace ohledně VOIP a web kamery.



Obrázek 8: Nabídka nastavení

Nakonec přidávám pohled na původní grafické rozhraní klienta v příloze č. 30. Rozvržení jednotlivých oblastí či pod-oken v okně je stejné nebo podobné. Grafické rozhraní každopádně nevypadá příliš přitažlivě a obsahuje nadbytečné nabídky, které byly v rámci nového rozhraní zahrnuty do jiných nabídek nebo vymazány pro nadbytečnost. Pro příklad byla nahrazena komponenta pro zobrazování prezentací modulů za modernější, přidána možnost posunovat s boxem zpráv, MVVM pro seznam přihlášených uživatelů nebo přizpůsobení dle role uživatele. Množství změn je každopádně obsáhlejší.

3 Původní zásuvné moduly

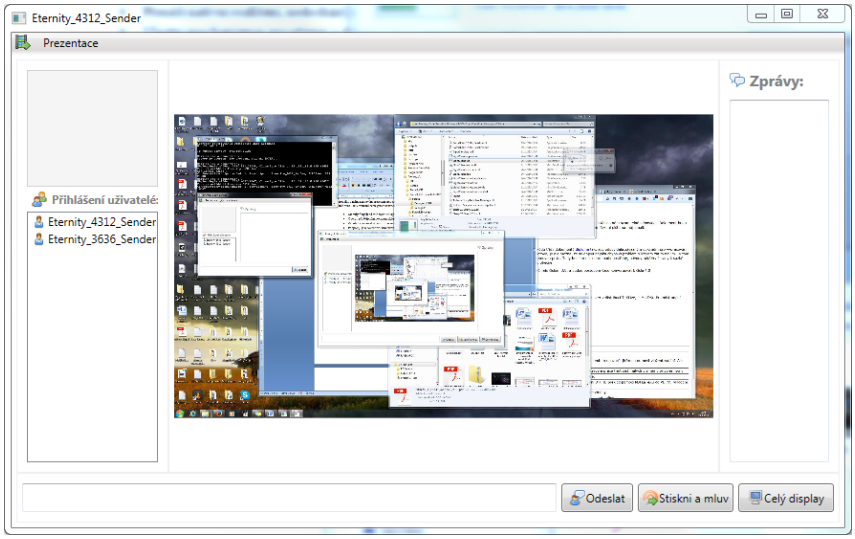
Původní zásuvné moduly obsažené v systému ForceB se věnují přenosu a sdílení obrazu. Konkrétně přenosu celé obrazovky uživatele nebo její části v podobě okna aplikace. K tomuto účelu využívali interně GDI+ technologii a její funkcionalitu. Mým zadáním bylo tyto moduly přepracovat do takové podoby, aby došlo k optimalizaci při práci s obrazem. Předtím než se ale budu věnovat provedeným optimalizacím a jejich implementaci a realizaci, chtěl bych stručně přiblížit původní moduly a jejich práci s obrazem.

3.1 Sdílení plochy

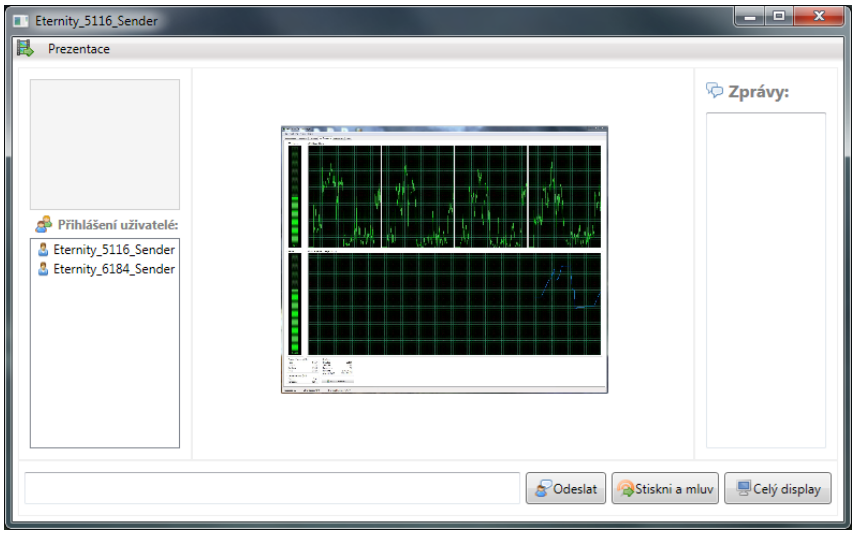
Modul umožňuje sdílet obraz celé plochy monitoru uživatele. Můžeme tak vidět vše, co se na obrazovce vyučujícího aktuálně děje. Toho se dá využít pro výuku programování, instruktáže jak něco nakonfigurovat nebo nastavit, pomoc při řešení chyb nebo jiného problému. Ukázku přenosu obrazu můžeme vidět na obrázku č. 9.

3.2 Sdílení okna aplikace

Modul je uzpůsoben pro sdílení konkrétního okna aplikace a jeho podoken, které jej překrývají. Uživateli v roli učitele je po spuštění modulu zobrazen dialog pro výběr aplikace, kterou chce spustit jako sledovanou. Učitel tak může schovat zbytek plochy před studenty a soustředit se jen na práci s konkrétní aplikací. To se hodí pro tutoriály ukazující práci s konkrétním softwarem. Příkladem může být 3D modelovací software, dále software pro vývojáře jako Visual Studio nebo jiný grafický nebo hudební software. Obrázek č. 10.



Obrázek 9: Modul pro sdílení plochy



Obrázek 10: Modul pro sdílení okna aplikace

3.3 Původní práce s obrazem

Původní práce s obrazem v obou výše zmíněných modulech probíhá v následujících fázích, které si dovolím stručně přiblížit.

1. Získání snímku jako objektu typu Bitmap.
2. Vyhledání změn ve snímku a jejich extrakce a přenos.
3. Příjem obrazových dat, jejich sestavení zpět na Bitmap a vykreslení.

Fáze 1 Původní moduly využívají pro přenos obrazu starší rozhraní IPresentationCamWPF, které v té době bylo moduly implementováno. To bylo navrženo pouze pro přenos obrazových dat a s širším použitím nepočítalo. Když uživatel spustil modul, aplikace jej zavedla a zaregistrovala si při tom odběr události NewFrame ze jmenovaného rozhraní. Událost můžeme vidět v kódu rozhraní.

```
public interface IPresentationCamWPF
{
uint SSRC { get; set; }
PresentationBasePage fPresenationPage { get; set; }
System.Drawing.Size fResolution { get; set; }
void PresentationPage_ControlBuffer(object sender, PresentationControlEventArgs
    e);
bool Start();
void SignalToStop();
void WaitForStop();
event NewFrameCamEventHandler NewFrame;
}
```

Výpis 1: Kód původního rozhraní

Konkrétní modul pro práci s obrazem začne po spuštění odebírat snímky skrze GDI+ pomocí třídy ScreenCapturing. Třída obsahuje sadu metod pro práci s obrazem využívající objekt typu Bitmap. Bitmap v NET Frameworku zastupuje totiž zmíněné GDI+ a slouží jako jakýsi wrapper obalující jeho funkcionalitu. Pro získání obrazu tak zavoláme jen příslušnou metodu nad třídou ScreenCapturing. Metoda zpět předá snímek reprezentovaný jako Bitmap a ten je následně předán události NewFrame. Příklad volání metody ze třídy ScreenCapturing můžeme vidět níže.

```
var bmp = ScreenCapturing.GetPrimaryDesktopWindowCaptureAsBitmap();
NewFrameCamEventArgs nfpea = new NewFrameCamEventArgs("", bmp);
if (NewFrame != null)
   NewFrame(this, nfpea);
```

Výpis 2: Použití třídy ScreenCapturing

Fáze 2 Jakmile aplikace tímto způsobem obdrží snímek, je zpracován třídou PresentationCamEnumerator zajišťující odběr dat ze zmíněné vlastnosti NewFrame. Třída si odebere předaná data a zpracuje je pomocí třídy SquareCompression. Třída SquareCompression zde slouží k nalezení změn v obraze. Nejprve je vyvolána metoda CheckBitmap, která provede nalezení změn ve snímku oproti předchozímu. Metoda nám zpět vrátí list změn. Pokud jsou v listu zaznamenané nějaké změny, jsou předány metodě PictureBlocksToArray. Ta provede finální extrakci změn do pole bytů. Pole je následně předáno zpět třídě PresentationCamEnumerator a ta jej odešle sítí. Použití třídy SquareCompression lze vidět níže.

```
List<PictureBlock> pictureBlocks = squareCompressions.CheckBitmap(bmp,
    AppSettings.Default.SendFullSize,AppSettings.Default.SCRatioBitMaps,
    AppSettings.Default.SCCheckPixel, AppSettings.Default.SCSendFullSizeEvery,
    AppSettings.Default.ViewChanges);

if (pictureBlocks.Count == 0)
   return;
buffer = squareCompressions.PictureBlocksToArray(bmp.Width, bmp.Height,
    pictureBlocks, AppSettings.Default.PresentationCamQuality);
```

Výpis 3: Použití třídy SquareCompression pro nalezení změn

Fáze 3 Při příjmu dat na straně studenta aplikace zavede modul, kterému jsou data určeny. Pokud je již zaveden, tak mu data jen předá. Zároveň zaregistruje modulu příjem dat skrze třídu CamShow. CamShow příjme data předaná ze sítě jako pole bytů a převede je zpět na list změn, ty následně vykreslí do finálního snímku. Pro převod pole bytů na list změn voláme znovu třídu SquareCompression a její metodu ArrayToPictureBlocks.

```
List<PictureBlock> pictureBlocks = squareCompressions.ArrayToPictureBlocks(
    buffer, out width, out height);
```

Výpis 4: Použití třídy SquareCompression pro opětovné získání změn z pole bytů

Tímto původním způsobem zpracování projdou postupně všechny snímky a právě tento proces nahradím novým.

3.4 Optimalizace zpracování obrazu

Dřívější zásuvné moduly Sdílení Plochy a Sdílení Okna Aplikace se jako jediné věnují celému procesu práce s obrazem. Tudíž právě u nich jsem se zaměřil na optimalizaci celé jejich práce od začátku až do konce. Nový proces zpracování obrazu se skládá z několika fází:

1. Získání snímku plochy či aplikace.
2. Porovnání aktuálního snímku oproti předchozímu s nalezením následných změn v obraze a jejich extrakcí z aktuálního snímku.

3. Zakódování všech jednotlivých změn do JPEG formátu.

4. Zabalení změn do finálního kontejneru a jeho následná serializace do binárního formátu a odeslání skrze síť.

5. Příjem binárních dat, jejich deserializace zpět na objekt kontejneru obsahujícího extrahované změny.

6. Vykreslení jednotlivých změn z kontejneru do aktuálního snímku a jeho následné zobrazení uživateli.

Tento proces se opakuje s každým dalším získaným snímkem v bodě jedna. Je třeba si uvědomit, že takto zpracujeme až 30 snímků za jednu sekundu. Jde tedy o kritickou oblast, kde hraje hlavní roli rychlost kódu v jednotlivých částech procesu. Pokud by totiž některá část procesu nadměrně zdržovala ostatní části, vedlo by to ke zpomalení celého procesu nebo až k jeho úplnému selhání. Takovouto část bychom nazvali úzkým hrdlem a rychlost provedení celého procesu by se logicky omezila na rychlost této části. Každopádně se dá předpokládat, že úzkým hrdlem bude vždy propustnost síťové linky, jelikož vždy bude záležet na tom, kolik obrazových dat za sekundu vyprodukujeme a budeme tak chtít odeslat. Množství vyprodukovaných a tedy odeslaných dat lze samozřejmě snížit pomocí komprese nad daty, k tomuto účelu využíváme JPEG formát. JPEG formát nabízí z tohoto pohledu hned několik výhod a to ztrátovost a vysokou kompresi, která se ztrátovostí přímo souvisí. Další modifikací, která snižuje množství přenesených dat, je odesílání změn v obraze namísto celých částí obrazu. Každopádně vždy bude vše záviset na propustnosti síťové linky, protože i při snížení velikosti dat výše popsanými technikami, nebudeme nikdy znát reálnou propustnost uživatele.

Při hledání slabých částí v celém procesu zpracování obrazu jsem se zaměřil hlavně na viditelně pomalé části, části s neefektivním kódem, ale také na dříve použitou technologii GDI+. Viditelně pomalou částí bylo například získávání jednotlivých snímků za sekundu, kdy použitá technologie GDI+ jednoduše nestačila efektivně plnit naše požadavky. Dále jsem nalezl nemalé množství kódu, který jsem vyhodnotil jako časově, paměťově či výpočetně neefektivní. Takovýto kód se vyskytoval zvláště v kritických místech samotného procesu zpracování, jako získání snímku, nalezení změn ve snímku oproti předchozímu, kódování a dekódování snímku do formátu JPEG a vykreslení snímku do prezentace uživateli. Spousta těchto výše popsaných kritických míst prováděla zbytečné alokace paměti, kopírování či konverzi dat, které nebyly potřeba a daly se nahradit jiným způsobem. Na základě všech těchto problémů jsem byl jednoduše nucen vše implementovat nanovo s důrazem na vysokou úroveň optimalizace kódu.

3.4.1 Nahrazení původního rozhraní GDI+

Jak bylo zmíněno výše, GDI+ rozhraní nedovede zajistit dostatečný a rovnoměrný výkon na všech počítačích. To je zapříčiněno jen částečnou podporou hardwarové akcelerace a také pozdějším pokusem o nahrazení pomocí technologie Direct2D ve Windows 7 a Windows Vista, který

se zdá být nakonec úspěšným. Navíc samotné GDI+ je jakýmsi rozšířením a vylepšením původního GDI rozhraní, které nabízelo prakticky plnou hardwarovou akceleraci a v mnoha ohledech bylo výkonnější. Tuto původní technologii jsem se rozhodl nahradit technologií, která přistupuje přímě ke GPU a má tak plnou hardwarovou akceleraci. Zvolil jsem proto DirectX technologii, která přesně tyto možnosti nabízí a zároveň se nám tak podaří některé výpočty či operace přesunou na GPU, čímž zvýšíme výkon CPU, který byl dříve mnohem více zatížen skrze GDI+ řešení.

3.4.2 Limitace DirectX ve Windows

Ačkoliv se zdá, že řešení skrze DirectX nám umožňuje tímto vyřešit všechny problémy, není to tak docela pravda. DirectX je rozhodně velmi výkonné řešení pro zachycení obrazu a případné zpracování, ale je třeba si uvědomit, že zde hraje roli i verze daného Windows OS. Každá verze operačního systému Windows vždy přišla zpravidla s úplně novou verzí tohoto rozhraní. Jednotlivé verze operačního systému Windows tedy využívají odlišnou verzi DirectX technologie, jak můžeme vidět v tabulce 1.

Tabulka 1: Jednotlivé verze rozhraní DirectX

Verze Windows OS (Desktop)	Verze DirectX (nejvyšší podporované)
Windows XP	DirectX 9c
Windows Vista, SP1, SP2	DirectX 10 a 10.1, 11 třeba instalovat update
Windows 7, 7.1	DirectX 11, 11.1 pouze částečně
Windows 8	DirectX 11.1
Windows 8.1	DirectX 11.2
Windows 10	DirectX 11.3 a 12

Nové verze operačního systému Windows vždy obsahují předchozí verze DirectX rozhraní pro zpětnou kompatibilitu. To umožňuje aplikacím využívajícím starší verze DirectX rozhraní bezproblémově běžet.

3.4.3 DirectX a optimální řešení

Zmíněné faktory jako kompatibilita a jednotlivé verze DirectX rozhraní nás částečně limitují. Dalším problémem je, že až od Windows 8 je možné přímě přistupovat k front bufferu a brát z něj data. Front buffer je část paměti, která je zpracována grafickým adaptérem (GPU) a následně zobrazena na monitor. Aplikace zde nikdy nezapisují data. Zatímco back buffer je část paměti, která je využívána aplikací pro přímý zápis dat, tato část paměti není nikdy přímě zobrazena na monitor. Mezi těmito buffery probíhá přehazování obsahu, kdy je tímto způsobem vykreslen na monitor. Pro získání aktuálního snímku plochy je třeba tedy přistoupit k front bufferu a získat jeho obsah. Protože to je ale možné až od verze DirectX 11.1, můžeme takové řešení použít pouze od Windows 8. To nás staví do značně nevýhodné pozice a je docela nápadné, že unikátní

řešení skrze jedinou verzi DirectX rozhraní pravděpodobně neexistuje. Zbývá nám tedy použít obdobné řešení na starší verzi DirectX rozhraní nebo jinou alternativní technologii pro ostatní Windows verze operačního systému.

Alternativou by bylo získat přístup přímo na rozhraní DirectX aktuální plochy Windows, protože Windows samotný také používá DirectX k vykreslování svého obsahu. Takové řešení by ale vyžadovalo injekci našeho kódu do části Windows systému. To by bylo hodně obtížné, každopádně se ukázalo, že i realizovatelné. Proces, který ve Windows systému spravuje naši aktuální plochu, se jmenuje dwm.exe, ve zkratce to znamená Desktop Window Manager, jiným názvem Správce oken plochy. DWM proces se objevil poprvé s příchodem Windows Vista a zajišťuje kompletní vykreslování a efekty pro Windows Aero funkcionalitu, která se objevila taktéž s příchodem výše jmenovaného operačního systému. Každopádně proces dwm.exe obsahuje pravděpodobně nějaký druh ochrany proti injekci jeho DirectX rozhraní a tudíž i po úspěšném získání rozhraní je reference na toto rozhraní po určité době zneplatněna a je třeba provést celý proces injekce znovu, což tento typ možného přístupu značně znehodnocuje. Navíc každá verze Windows, a to počínaje od Windows Vista, má zpravidla nějak upravený dwm.exe proces.

Další alternativou, na kterou jsem narazil, je velmi nekonvenční přístup, který k získání rozhraní DirectX, které aktuálně daný Windows systém využívá, použije privátní metody z jádra Windows. Funkcionalita těchto knihoven nebyla zdokumentována a je zcela jisté, že Microsoft nechtěl, aby se dalo tímto přístupem získat či manipulovat s tímto rozhraním. Toto řešení je vlastně druh hackingu, který dokonce dovoluje vykreslovat nám přímo do rozhraní Windows vlastní obsah.Tuto variantu uvádím spíše ze zajímavosti, protože by její použití bylo v rozporu s Windows licencí.

Nakonec jsem nalezl možnost využít část Windows funkcionality, abych na nepodporovaných Windows verzích získal obraz se srovnatelným výkonem jako DirectX. Nalezená knihovna MagnifierAPI podporovaná od Windows Visty je dnes na všech Windows systémech. Knihovna interně využívá GDI a hlavně DirectX. To samo o sobě značí velmi vysoký výkon. Nejzajímavější je, že tato knihovna a její rozhraní nebylo ani není pro získávání obrazu navrženo. Její funkcionalita nabízí Windows komponentu lupy pro přibližování obrazu. Po počátečních problémech se mi ale podařilo vše zprovoznit a zastoupit tak DirectX na nepodporovaných OS systémech tímto řešením.

3.5 Algoritmus pro vyhledávání změn

Následující popisovaný algoritmus je součástí komponenty pro vyhledávání změn v obraze. Jde o plně paralelizovatelný algoritmus, který dovede běžet jak synchronně, tak plně paralelně. Stejně tak se umí přizpůsobit jakémukoli rozlišení porovnávaných snímků, kdy rozlišení porovnávaných snímků musí být logicky totožné. Logicky nedává smysl porovnávat snímky o rozdílném rozlišení. Zároveň je schopný detekovat změnu v rozlišení za běhu. Tím máme na mysli to, že pokud algoritmu za běhu předáme snímek o jiném rozlišení, je detekce přeskočena do doby, dokud není rozlišení obou porovnávaných snímků totožné. Zpracování obrazu probíhá na low-level úrovni,

kdy se provádí výpočty nad samotnými byty v obraze. Algoritmus je zároveň schopný zajistit 100% detekci změn v obraze s přesností na jeden jediný pixel. Tedy pokud dojde ke změně pouze v jednom jediném pixelu, bude tato změna detekována úspěšně. Algoritmus taktéž při nalezení změn provádí i extrakci změn z důvodu, aby operace extrakce a kódování změn využily paralelní běh. Uvedené vlastnosti a mnohé další jsou nastavitelné na základě předaných parametrů a je možné je měnit za běhu. Samotný popis a použití komponenty je obsažen v sekci MultiCapturer. Pro dosažení paralelismu využíváme Task Parallel Library od společnosti Microsoft určenou pro NET platformu.

3.5.1 Průběh a fungování

Algoritmus rozdělí snímek dle nastavených parametrů horizontálně na tzv. vrstvy a vertikálně na tzv. areály. Tím získáme v každé vrstvě x areálů, kdy tyto areály do této vrstvy pevně patří. Pokud není rozlišení porovnávaných snímků dělitelné počtem požadovaných areálů či vrstev, vznikají nám tzv. částečné areály či tzv. částečná vrstva. S těmito částečnými areály či částečnou vrstvou algoritmus logicky předem počítá. Výpočet na jednotlivých vrstvách lze paralelizovat, tedy každá vrstva může být zpracována paralelně a nezávisle na ostatních vrstvách. Z toho vyplývá, že vrstva je nejmenší paralelizovatelnou jednotkou v rámci porovnávání jednotlivých snímků. Při paralelním výpočtu bychom měli využít počet fyzických či logických procesorových jader na daném počítači k dosažení optimálního výkonu. To však není nutnou podmínkou, jelikož je každá vrstva paralelizovatelná, je proto možné použít pro každou vrstvu jedno samostatné vlákno. Tím ale nemusí dojít k optimální paralelizaci, protože při více vláknech musí být vlákna zpracována menším počtem jader. Je tedy žádoucí, aby počet vláken zastupujících jednotlivé vrstvy odpovídal počtu jader, jak bylo zmíněno výše.

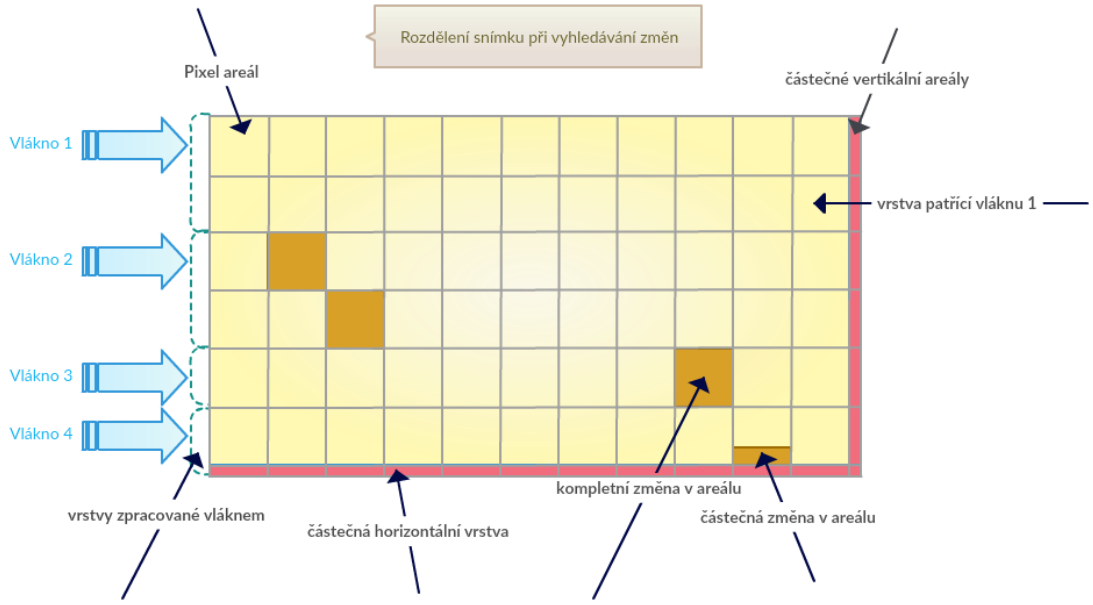
Vrstvy jsou před zpracováním přiděleny jednotlivým procesorovým jádrům, která jsou v operačním systému zastoupeny vlákny. Jednotlivá vlákna zastupující jádra CPU provedou následné paralelní zpracování. Každé vlákno zpracovává přiřazené vrstvy od jedné do n-té vrstvy, tedy postupně. Areály v každé vrstvě jsou taktéž zpracovány od jednoho do n-tého, taktéž postupně. Jakmile jsou všechny areály ve vrstvě zpracovány, přesune se vlákno na další přidělenou nezpracovanou vrstvu v pořadí. Tímto způsobem se zpracují všechny areály ve všech vrstvách daného vlákna.

Zpracování samotného areálu v aktuálním snímku probíhá porovnáním všech jeho řádků oproti všem řádkům areálu předchozího snímku. Zpracování řádků probíhá od prvního do n-tého řádku. Při nalezení změny na řádku jsou data areálu od pozice tohoto řádku obsahujícího změny extrahována až po n-tý poslední řádek areálu. Tato extrakce se provádí logicky pro areál z aktuálního snímku, kde byla nalezena změna oproti předchozímu snímku, jelikož aktuální snímek obsahuje naše data k extrakci.

Oproti předchozímu řešení postavenému nad technologií GDI+, dokáže nový algoritmus v určitých situacích značně snížit velikost extrahovaných změn. A to právě díky tomu, že změnu začne extrahovat až od řádku, ve kterém změnu objevil, tedy řádky, které se nacházejí před

tímto řádkem se změnou jsou ignorovány. Tento postup by se dal dále modifikovat, aby dokázal takovou aplikaci i směrem ze spodu areálu směrem nahoru. Každopádně aktuální řešení tuto další modifikaci neimplementuje, protože po nalezení řádku obsahujícího změnu, nemusí další řádky ověřovat na změny, ale jednoduše je extrahuje také, což výrazně zvyšuje rychlost řešení. Extrahovaná data obsahující změny jsou dále zakódována do JPEG formátu a následně vložena do objektu kontejneru obsahujícího změny pro aktuální snímek.

Tímto způsobem jsou postupně zjištěny všechny změny v obraze, algoritmus před koncem počká, až přidělenou práci dokončí všechny vlákna prezentující jádra na CPU. Jelikož mohou některá vlákna dokončit práci dříve než ostatní, je třeba vždy počkat na všechna. Zmiňovaný kontejner uchovává vždy změny pro aktuální snímek. Tento kontejner je následně předán ke zpracování dále. Pro snadnější porozumění jak algoritmus operuje přidávám ilustraci č. 12.



Obrázek 11: Zobrazení rozdělení snímku při vyhledávání změn s použitím 4 vláken

Na snímku v příloze č. 29 můžeme vidět, jak v realitě algoritmus pracuje. Černé areály reprezentují plochy pixel areálů beze změny, zbytek viditelných částí odlišných od černých areálů reprezentuje již přenesené a zobrazené změny v obraze. Pro lepší znázornění jsem označil rohy černých areálů červenými body. Snímek níže tedy zobrazuje, která data se k nám přenesly a promítly a která naopak ne.

3.5.2 Další optimalizace

Kromě již zmíněné optimalizace, která přispěla ke snížení celkové velikosti extrahovaných areálů, byly provedeny i další významné modifikace, zvláště zrychlující samotný běh algoritmu. Aby všechny kritické operace běžely co nejrychleji, byla využita interní funkcionalita Windows knihoven psaných v C/C++ jazyku, kdy pro vyvolání této funkcionality je použito P/Invoke

(Platform Invoke) volání. Platform Invoke je část NET funkcionality, která dovoluje volat nativní neřízený kód z knihoven psaných v jiném jazyce než z rodiny NET jazyků. To se týká hlavně kopírování dat a porovnávání dat. Pro kopírování dat není použita funkcionalita NET Frameworku, ale systémová funkce memcpy. Ta je dostupná na všech Windows systémech a patří mezi nejrychlejší funkce pro kopírování dat. Její rychlost spočívá zvláště v tom, že pro kopírování využívá 128bit instrukce procesoru a její kód je psaný v assembleru a tudíž vysoce optimalizovaný. Naopak pro porovnávání dat využíváme memcmp funkci, která je taktéž velmi efektivně implementovaná v rámci Windows OS. Použití těchto dvou zmíněných funkcí, namísto funkcionality NET Frameworku, je zapříčiněno hlavně tím, že většina kódu, který se volá, se opakuje v určitých intervalech. Ušetření několika milisekund či mikrosekund v každé iteraci či intervalu dokáže výrazně ušetřit a srazit celkový čas pro provedení celkového kódu. To platí zvláště když jsou takové části volány často či periodicky. NET Framework sice nabízí hned několik funkcí na kopírování dat, každopádně jejich výkon není vhodný pro aplikace, které pracují v reálném čase. Ačkoliv některé z jeho funkcí volají přímo C/C++ funkcionalitu a tedy nativní kód, mohou provádět interní ověření parametrů a podobně, což degraduje malou měrou výkon. Dalším důvodem je i to, že při našem volání funkcí memcpy a memcmp využíváme speciální atribut nad těmito funkcemi, a to SuppressUnmanagedCodeSecurity. Tento atribut dovoluje nativnímu kódu volat řízený kód v C# přímo, navíc nedochází ke kontrolám při marshallingu. Marshalling je proces přenosu a konverze dat z jednoho formátu do jiného, často využívající serializaci a deserializaci dat. Díky tomu můžeme dosáhnout zase o něco vyššího výkonu, a proto je třeba volat memcpy a memcmp přímě pomocí naší implementace.

3.5.3 Optimalizace ohledně kódování JPEG

Jak již bylo zmíněno na úvod, původní řešení pro zpracování obrazu v zásuvných modulech využívalo GDI+ k získávání jednotlivých snímků. GDI+ bylo využito i dále v rámci algoritmu na vyhledávání změn v obraze. Tento algoritmus nalezené a extrahované změny následně kódoval do JPEG formátu právě pomocí GDI+ funkcionality. Kódování skrze GDI+ není sice vyloženě nejpomalejší, ale zároveň není ani nejrychlejší nebo nejefektivnější v poměru kvality ku kompresi. Protože jsem hledal slabá místa, označil jsem kódování jako jedno z těchto míst. To zvláště proto, že obrazová data přenášená skrze síť musejí být co nejmenší, což právě JPEG formát splňuje. Každopádně proces kódování do JPEG formátu se skládá z mnoha fází a je tedy poměrně náročný a složitý, právě to je jedním z argumentů pro použití co nejefektivnějšího JPEG kodéru. Dalším argumentem je to, že nepřenášíme celý snímek zakódovaný v JPEG formátu, nýbrž spoustu malých extrahovaných částí, kdy každou část je třeba kódovat zvlášť. Stačí se podívat ke kolika změnám dochází, když uživatel pohne kurzorem myši. Většina položek pod kurzorem nějak reaguje a mění obsah. Ještě horší situace nastane, začneme-li pohybovat s různými okny aplikací či otevírat různé nabídky nebo posunovat obsah v internetovém prohlížeči. Úplně nejhorší situací je, když uživatel přehrává nějaké video, animaci, která zabírá většinu nebo i celou obrazovku. V takovém případě dochází ke změnám prakticky po celé obrazovce, což zapříčiní detekci velkého

množství změn. Velké množství změn dále zapříčiní řetězovou reakci pro další body procesu zpracování obrazu. Dojde k větším datovým přenosům a náporu na síťovou linku a bude třeba dekódovat mnohem větší objem dat na druhé straně u klienta. V neposlední řadě bude třeba vykreslit všechna příchozí data, což se také projeví zvýšeným náporem. Některé zmíněné operace vyvolají velké množství změn v obraze, které je třeba následně zakódovat co nejrychleji. Právě proto je potřeba mít rychlý kodér pro JPEG formát.

Při hledání alternativy pro nahrazení GDI+ kódování jsem se zaměřil nejprve na modernější a dnes běžně používaný WPF Framework, který většinu funkcionality z WinForms nahradil. WinForms aplikace využívají ke svému vykreslování právě GDI a GDI+ a staví celkově na klasickém přístupu vývoje Windows aplikací. Zatímco WPF Framework vykresluje aplikace pomocí DirectX rozhraní. WPF aplikace tak nabízejí lepší výkon a využívají akceleraci nad GPU, zároveň dovolují použití 2D i 3D efektů přímo v okně aplikace.

3.5.4 Windows Imaging Component

WPF Framework využívá pro kódování a dekódování JPEG formátu interně WIC komponentu. WIC je Windows komponenta nabízející API ohledně práce s obrazovými formáty a jejich metadaty. Umožňuje kódování a dekódování různých formátů, správu barev, podporu JPEG YCbCr formátu a editaci metadat, zároveň nabízí rozšiřitelnou architekturu pro nové kodeky a také konverzní framework. WIC je o něco rychlejší než GDI+ pro kódování, ale rozdíl v rychlosti kódování není zase tak obrovský. Jedná se tedy o něco lepší variantu. Proto jsem se dále zaměřil na knihovny třetích stran. Ukázalo se ale, že žádná volně dostupná knihovna pro NET Framework nenabízí výkonnější řešení než WIC nebo GDI+ rozhraní.

3.5.5 Libjpeg

Jasnou volbou se tedy staly C/C++ knihovny, které by bylo možno volat pomocí P/Invoke volání nebo C++/CLI wrapperu přímo v C#.NET. Klasickou knihovnou v C/C++ pro práci s JPEG formátem je volně dostupná libjpeg, což je knihovna vyvinutá v roce 1991 skupinou Independent JPEG Group, která ji nabízí pod BSD licencí. Další nespornou výhodou je, že je poměrně dobře optimalizovaná díky tomu, že je vyvíjena až dodnes, což s sebou přináší podporu v podobě opravy chyb a nových verzí. Za zmínku také stojí fakt, že se jedná o open-source knihovnu, její kód je tedy volně dostupný v rámci BSD licence. Tako knihovna se těší velké popularitě mezi vývojáři softwaru, jelikož má mnoho odvozených verzí s novou dodanou funkcionalitou či jinými modifikacemi. Také existují její speciálně optimalizované verze, které dosahují výrazně vetší účinnosti během kódování a dekódování obrazových dat. Libjpeg tedy není nejvýkonnějším řešením na trhu, tím je jeho alternativa v podobě libjpeg-turbo.

3.5.6 Libjpeg-turbo

Libjpeg-turbo je jedna z nejvýkonnější volně dostupných open-source knihoven pro práci s JPEG formátem. Je založena nad libjpeg knihovnou a využívá SIMD instrukce procesoru k plné akceleraci výkonu, dále obsahuje vysoce optimalizované Huffmanovo kódování. Její prvotní verze byly založeny nad projektem libjpeg/SIMD, později v roce 2009 nad TigerVNC a VirtualGL projekty, které provedly výrazné vylepšení v kodeku. V roce 2010 došlo k založení nezávislého projektu, jehož cílem bylo vytvořit vysoce výkonnou JPEG knihovnu pro kompresi a dekompresi dat, která by byla volně dostupná většímu množství uživatelů a vývojářů. Ačkoliv je knihovna určena pro C/C++ vývojáře, brzy se stala dostupná také jako Java wrapper, pod jménem Turbo-jpeg. Navíc je multiplatformní a je tedy použitelná prakticky na všech dnešních systémech. Multiplatformnost samozřejmě znamená, že na každé platformě je třeba využít jiné SIMD instrukce, knihovna podporuje instrukce typu SSE, SSE2, MMX, NEON, AltiVec a další. Její vývojáři se pyšní tím, že je 2x až 6x rychlejší než původní libjpeg, ze kterého je také odvozena. Rychlost je taková, že je dokonce rivalem vysoce rychlých proprietárních řešení jako Intel® IPP. Také je součástí mnoha velmi známých programů jako Mozilla Firefox, Google Chrome, LibreOffice, Amazon Cloude Drive nebo dokonce operačních systémů jako CentOS, Debian, Fedora, Red Hat Enterprise, Ubuntu a dalších linuxově založených systémů a distribucí. Za zmínku ještě stojí někteří ze sponzorů, kteří vývojáře podpořily jako Mozilla, Google, IBM, AMD a další velcí hráči na trhu.

3.5.7 turbojpegCLI wrapper

Abych mohl libjpeg-turbo použít přímo z NET Frameworku, bylo třeba vytvořit či mít nějaký wrapper nebo P/Invoke funkcionalitu pro volání libjpeg-turbo API. Zjistil jsem, že existují pro NET Framework dva různé wrappery zprostředkující volání na libjpeg-turbo API. Jeden implementovaný interně pomocí P/Invoke volání a druhý nazvaný turbojpegCLI implementovaný pomocí C++/CLI. Druhý byl dle mého uvážení výhodnější volbou, jelikož C++/CLI má oproti P/Invoke volání značné výhody. C++/CLI je obdoba C++ jazyka určeného pro NET platformu, dokáže pracovat jak s řízeným, tak neřízeným kódem. Díky tomu je ho možné použít z jakéhokoli NET jazyku, to přináší značné výhody co se týká použití. Taktéž můžeme přistupovat k paměti spadající pod Garbage Collector a zároveň k nativní paměti. Zmíněný wrapper ale nebyl pro mé použití úplně optimálně implementovaný. Bylo tedy třeba změnit a dopsat část nové funkcionality přímo do tohoto wrapperu, jinak bych nemohl libjpeg-turbo API vůbec využít. Bylo by jej možné použít, ale tím bych přišel o jednu výraznou optimalizaci, tj. o přímý zápis a kódování dat do určeného bufferu v jediném kroku. Původní wrapper totiž umí pracovat jen s řízenou pamětí spravovanou Garbage Collectorem. Jenže má komponenta zaznamenávající obraz ukládá vše do speciálního typu bufferu, který přebývá v nativní paměti. Ve dřívější implementaci jsem toto vyřešil zavedením mezipaměti, která byla řízená. Tato mezipaměť byl vlastně druh bufferu, kam bylo třeba překopírovat data a teprve následně volat wrapper pro jejich zakódování. Pro-

tože byla mezipaměť řízená, wrapper vše vykonal úspěšně. Je ale viditelné, že bylo třeba provést nejprve zcela zbytečné překopírování dat do mezipaměti a poté zakódování do JPEG formátu. Proto jsem se později rozhodl tuto část přepsat pro již zmiňovaný přímý přístup. Hlavním důvodem pro mě bylo zachovat konzistenci u všech optimalizací, nechat zde původní řešení by bylo nedůsledné. TurbojpegCLI je naštěstí open-source pod zjednodušenou BSD licencí a tudíž nebyl problém provést potřebné modifikace. Nová verze mi nově umožnila kódovat jak řízenou, tak neřízenou paměť. Zatímco dřívější verze si poradila pouze s řízenou pamětí, se kterou jsem ale vůbec nepracoval. Práce s neřízenou pamětí byla zapříčiněna tím, že moje komponenta pro získávání snímků plochy či aplikace ukládá data do bufferů, které jsou alokovány jako nativní paměť. Buffery jsou totiž systémovými objekty v rámci Windows, konkrétně Memory Mapped Files objekty. Jde znovu o typ optimalizace, který je blíže popsán dále v textu v kapitole ohledně MultiCapturer knihovny a její implementace. Jelikož ke kódování nalezených změn dochází v algoritmu pro vyhledávání změn, bylo třeba pouze nahradit dřívější GDI+ kódování voláním nově vytvořené implementace turbojpegCLI wrapperu.

3.5.8 Injekce IL a realtime zavedení DLL

Zde nastal ale další problém, a sice že turbojpegCLI wrapper volá libjpeg-turbo C++ implementaci. Jelikož je libjpeg-turbo napsaný v C++, obsahuje zvlášť verzi knihovny pro platformu x86 a x64. Jenže veškerá zmíněná funkcionalita ohledně získání obrazu, nalezení změn, kódování a dekódování byla mnou implementována v rámci .NET knihovny MultiCapturer, kterou využívají zásuvné moduly pracující s obrazem. Tako knihovna byla koncipována a vyvíjena od začátku jako AnyCPU, tedy s podporou x86 i x64 platformy. Bylo tedy třeba provést automatizaci pro zavádění správné verze libjpeg-turbo knihovny a konkrétního wrapperu pro tuto verzi. Wrapper pro x86 je linkován proti x86 nativní verzi libjpeg-turbo knihovny a wrapper pro x64 je linkován proti x64 nativní verzi libjpeg-turbo knihovny. Řešením je detekovat na jaké platformě běžíme a zavést správnou verzi wrapperu, proti které je nalinkován. Ačkoliv se řešení může zdát jednoduché, je mnohem těžší ho dosáhnout, než se může na první pohled zdát. To je dáno zvláště tím, že interně v MultiCapturer knihovně využíváme wrapper pro kódování a proto se na něj musíme v rámci projektu odkazovat. Jenže v rámci projektu se odkazujeme na jednu konkrétní zvolenou verzi, v našem případě x64. Pokud by byla ale knihovna MultiCapturer zavedena na platformě x86, vše by spadlo, protože libjpeg-turbo v odkazované verzi x64 by nebylo možné zavést pod x86 platformou. Pro dosažení výsledku jsem potřeboval již při inicializaci MultiCapturer knihovny provést detekci a zavést správnou verzi wrapperu. Toho jsem docílil tím, že jsem injektoval IL kód MultiCapturer knihovny, kdy jsem do její inicializace vložil kód autodetekce platformy a následného zavedení wrapperu. Toho jsem dosáhl pomocí Nuget balíku jménem InjectModuleInitializer určeného pro Visual Studio. Nápad jsem dostal na základě toho, jak fungují C/C++ knihovny, které obsahují inicializační metodu DllMain, něco jako ekvivalent Main metody v aplikacích. Podobná funkcionalita existuje interně i v NET Frameworku a nazývá se modul inicializér, každopádně není normálně dostupná. Modul inicializér

je spuštěn před jakýmkoli jiným kódem v daném modulu, navíc je automaticky spuštěn při zavedení knihovny a to pouze jednou. Tedy pokud zavoláme nějakou funkcionalitu z MultiCapturer knihovny, bude nejprve NET Framework zavádět naši knihovnu MultiCapturer do paměti. MultiCapturer se pokusí zavést své závislosti. Nalinkovaný wrapper pro platformu x64 ale nenajde, protože není spolu s knihovnou ve stejné složce a zaváděcí engine selže. Je to díky tomu, že jsem odkazovanému wrapperu pro platformu x64 nastavil vlastnost CopyLocal v rámci Visual Studia na false, to zapříčiní, že nebude wrapper automaticky kopírován do složky, kde se nachází MultiCapturer. To následně vyvolá událost AppDomain.AssemblyResolve na aplikační doméně MultiCapturer knihovny, kterou dále odchytíme a pokusíme se zavést knihovnu manuálně. Kód v události provede detekci v rámci jaké platformy běží proces využívající MultiCapturer a na základě ní jakoby manuálně zavede konkrétní wrapper pro danou platformu. Jakmile je wrapper zaveden, je následně zavedena knihovna libjpeg-turbo, kterou wrapper linkuje. Tím je docíleno plné podpory AnyCPU nad knihovnou MultiCapturer. Nutno dodat, že celý proces je spuštěn pouze pokud zavoláme nějakou funkcionalitu z MultiCapturer knihovny, tím totiž spustíme zavedení knihovny a dále zavedení jejich závislostí, tedy ostatních knihoven.

Volba libjpeg-turbo mi tedy v tomto ohledu umožnila výrazně zvýšit výkon při kódování a dekódování jednotlivých extrahovaných areálů do JPEG formátu. To přispělo ke snížení času potřebného pro zpracování dat v tomto kroku a tedy i zrychlení celého procesu kódování a dekódování.

3.5.9 Přenos dat skrze síť

Jakmile jsou všechna data úspěšně extrahována a zakódována algoritmem pro vyhledání změn v obraze do kontejneru obsahujícího změny, jsou připravena na přenos sítí. Kontejner je serializován na pole bytů a následně odeslán klientovi skrze síť. Používáme tedy binární serializaci, kdy se data převedou na zmíněné pole bytů. Binární serializace je obecně rychlejší a serializovaná data zabírají malou velikost. Pro serializaci dat využíváme výchozí serializér v NET Frameworku, ačkoliv existují i výkonnější varianty třetích stran. Bylo by zde možné sice snížit čas serializace nebo deserializace pomocí rychlejší varianty třetí strany, ale to by nemělo tak velký dopad, protože výchozí serializér je dostatečně rychlý. Data jsou přenesena pomocí ForceB protokolu a následně deserializována zpět na objekt kontejneru.

V tomto konkrétním kroku jsme velmi závislí na samotné přenosové síti, o čemž jsem se zmiňoval již v úvodu této kapitoly. Kromě co nejnižšího množství dat k přenosu a jejich co nejmenší velikosti zde není možné docílit žádné úspory či nějakého zrychlení. Každopádně pro zaručení optimálního přenosu bez chyb a výpadků nebo zpomalení, můžeme stanovit doporučenou rychlost linky pro uživatele. Doporučená rychlost linky by se měla odvíjet od rozlišení obrazu a nastavení, při kterém je přenos realizován. Uvádím orientační tabulku pro nejpoužívanější rozlišení, samotné statistiky ohledně přenosu jsou zmíněny poslední v kapitole této práce.

Tabulka 2: Přenosová rychlost sítě na základě nastavení nahrávací komponenty

Rozlišení (nastavení profilu)	Doporučená přenosová rychlost sítě (v Mb/s)
2560x1440, 30fps (2K)	15Mb/s - 19Mb/s
1920x1080, 30fps (FullHD)	10Mb/s - 15Mb/s
1280x720, 30fps (HD)	8.5Mb/s - 10 Mb/s
menší rozlišení než HD, 30fps	$<$ 8.5Mb/s

3.5.10 Vykreslení jednotlivých snímků

Jakmile získáme kontejner s obrazovými daty, pokusíme se je vykreslit. Kontejner reprezentuje konkrétní jeden snímek. Pro vykreslování obsahu využíváme objekt typu WriteableBitmap, který je možné přepisovat novými obrazovými daty. WriteableBitmap je objekt vyvinutý speciálně pro WPF aplikace, umožňující vykreslování obsahu jako animace nebo vizualizace dat. Je navržený pro vykreslování vícera snímků za sekundu. Využívá dva buffery v systémové paměti. Back buffer obsahuje obsah, který bude vykreslen v příštím snímku. Front buffer obsahuje obsah, který je aktuálně zobrazen na obrazovce. Obsah front bufferu je totiž vždy vykreslovacím systémem promítnut na obrazovku. Pro vykreslování jsou použita dvě různá vlákna, UI vlákno pro vkládání nového obsahu do WriteableBitmap a vykreslovací vlákno, které obsah přebere a vykreslí. Vykreslovací vlákno je vždy vytvořeno pouze jedno na celou aplikaci, zatímco UI vláken je možné vytvořit uměle více. Na stránkách Microsoftu existuje jakýsi doporučený vzor jak s tímto objektem pracovat.

1. Zavolat Lock metodu k reservaci back bufferu pro updaty.
2. Získání pointeru na back buffer.
3. Zápis dat do back bufferu, nyní mohou zapisovat změny i další vlákna.
4. Zavolání metody AddDirtyRect k označení areálů, ve kterých došlo ke změně.
5. Zavolat Unlock metodu k vypuštění back bufferu a vykreslení obsahu na obrazovku.

Výše popsaný postup nám dává hlavně vetší kontrolu nad jednotlivými voláními, ačkoliv existuje alternativa v podobě metody WritePixels, která tento postup vlastně zopakuje s tím, že je vše automatické a uvnitř metody. Zajimavostí je, že volání Lock, Unlock, AddDirtyRect, BackBuffer nad WriteableBitmap objektem je možné pouze z UI vlákna, to je dáno tím, že pouze UI vlákno může vkládat nový obsah do back bufferu. Proto jen UI vlákno může zamknout či odemknout back buffer pro přepis. Jakmile UI vlákno tímto způsobem aktualizuje areály a signalizuje pomocí metody Unlock odemknutí back bufferu, vykreslovací vlákno zkopíruje změněné areály v back bufferu, které ohlásila metoda AddDirtyRect do front bufferu. Vykreslovací systém kontroluje tuto výměnu bufferů, aby předešel deadlokům na vláknech a překreslovacím artefaktům v obraze, jako je tearing.

WriteableBitmap samotný je napojený na komponentu Image skrze vlastnost ImageSource. Odpadá tak potřeba alokovat pro každý nový snímek vždy nový BitmapSource, který byl dříve přiřazen Image komponentě skrze vlastnost ImageSource, což bylo značně neefektivní. Při dřívějším řešení se alokovalo velké množství objektů a navíc docházelo ke konverzi Bitmap dat do BitmapSource objektů. Novým způsobem jsme schopni skrze WriteableBitmap vykreslovat efektivně a přímě. Nový dekodér libjpeg-turbo je totiž schopen dekódovat data přímo do back bufferu, kdy jsou později přehozena do front bufferu a vykreslena.

4 Nové zásuvné moduly

4.1 Rozhraní pro zásuvné moduly

Zvláště kvůli realizaci nových zásuvných modulů bylo třeba vytvořit nové rozhraní pro jejich zavádění a správu. Původní rozhraní využívaly pouze zásuvné moduly pro sdílení plochy a okna aplikace. Celé rozhraní tak bylo navrženo účelově pro přenos obrazu, jenže funkcionalita nových modulů přinesla nový obsah pro přenos. Třeba modul whiteboard přenáší čistě kontrolní a kreslící data. Stejně tak budoucí moduly mohou vyžadovat přenos jiného typu dat. To vše vedlo k vytvoření nového rozhraní pro zavádění modulů spolu s novým rozhraním pro aplikaci a nově vytvořenou třídou pro jejich kontrolu.

4.1.1 Vývoj modulů pro ForceB

Vyvíjet zásuvné moduly může každý programátor NET Frameworku, stačí znát rozhraní pro vývoj modulů a správně jej implementovat. Moduly jsou vždy realizované formou dll knihoven a ty v sobě nesou obvykle i všechny použité dodatečné zdroje. Dodatečnými zdroji mohou být obrázky, ikony, UI rozhraní modulu v XAML kódu, styly pro komponenty a další. Knihovny třetích stran použité modulem je nutné k našemu modulu přiložit, díky tomu mohou později jiné moduly využít dodané knihovny.

Každý modul musí implementovat povinné rozhraní IPlugin, které zobecňuje použití modulu. Rozhraní specifikuje metody pro spuštění a ukončení běhu modulu spolu s vlastnostmi pro přístup k IPluginHost hostiteli modulu a jeho UI rozhraní. Dále musí každý modul být označen atributem PluginMetadataAtribute, který definuje metadata modulu jako jeho interně použité jméno, ikonu a jméno zobrazované v UI klienta. Odběr dat ze sítě realizuje vývojář skrze IPluginHost rozhraní, to je po zavedení modulu aplikací předáno vlastnosti Host na IPlugin rozhraní. IPlugin rozhraní zde reprezentuje samotnou instanci modulu. IPluginHost nabízí vývojáři práci s několika typy datových proudů, které může odebírat ze sítě.

- **Control Data** - umožňují odesílat kontrolní data pro synchronizaci nebo jiná čistá data v binární formě, formát dat a jejich reprezentace je čistě na vývojáři modulu.
- **Frame Data** - umožňuje odesílat frame data, což jsou speciální motion jpeg data vyprodukovaná komponentou pro vyhledávání změn, určeno pro stálý tok dat většího rozlišení.
- **Slide Data** - použitý pro přenos motion jpeg dat a jednotlivých slidů, nemusí jít o data posílaná v pravidelném intervalu.

Stejně tak data produkovaná modulem může vývojář předat pomocí IPluginHost rozhraní příslušným metodám ke zpracování. Takovým způsobem využívá vývojář IPluginHost rozhraní k odběru a přenosu dat. Výhodou IPluginHost rozhraní je, že může v budoucnu definovat i další

užitečné akce pro moduly. Poslední věcí, kterou bude vývojář modulu potřebovat, je třída PresentationBasePage, ze které by měl odvodit své grafické rozhraní modulu. PresentationBasePage definuje výchozí komponentu pro odvození UI a zajišťuje možnost odběru akce Dispose nad UI rozhraním modulu. Jedině dodržením výše popsaného procesu můžeme vytvořit nový validní modul. Pro přehled uvádím kód obou zmíněných rozhraní.

```
public interface IPlugin
{
PresentationBasePage PresenationPage { get; set; } // GUI Pluginu
IPluginHost Host { get; set; }                     // Hostitel pro plugin
bool Start(bool immediateCallFromManager);
void SignalToStop();
void Close();
}
```

Výpis 5: Kód rozhraní IPlugin

```
public interface IPluginHost
{
// Message for the user of the application.
void Message(MsgInfoType msgType , string title, string msg);
/* Events - plugins can register them in start() method to receive the
   requested data that were already fetched for them as EventArgs */
event FrameDataEventHandler ReceiveFrameData;
event ControlDataEventHandler ReceiveControlData;
event SlideDataEventHandler ReceiveSlideData;
// Receive plugin data (FRAME)
event FrameDataEventHandler PluginFrameData;
// Receive plugin data (CONTROL)
event ControlDataEventHandler PluginControlData;
// Receive plugin data (SLIDE)
event SlideDataEventHandler PluginSlideData;
void RaiseSlideData(SlideDataEventArgs args);
void RaiseFrameData(FrameDataEventArgs args);
void RaiseControlData(ControlDataEventArgs args);
/* Handlers - plugins can send data via this handlers, data must be fetched
    into EventArgs. */
void ControlDataEventHandler(object sender, ControlDataEventArgs e);
void FrameDataEventHandler(object sender, FrameDataEventArgs e);
void SlideDataEventHandler(object sender, SlideDataEventArgs e);
}
```

Výpis 6: Kód rozhraní IPluginHost

Kompletní nové rozhraní a jeho architekturu s popisem jednotlivých tříd můžeme vidět na obrázku č. 21 v příloze.

PluginManager Zavaděč zásuvných modulů umožňuje zavádět a spravovat moduly. Spravuje seznam všech použitelných modulů ze složky aplikace. Umožňuje zavádět a startovat nové moduly, ukončovat je, vytahovat z nich metadata a vytvářet UI nabídku modulů pro aplikaci.

IPluginHost Reprezentuje rozhraní pro hostitele modulů. Hostitelem modulů máme na mysli klientskou aplikaci, ta pomocí tohoto rozhraní může jednotlivým modulů nabídnout část své interní funkcionality, kterou přes IPluginHost vystaví. Moduly poté mohou funkcionalitu tohoto rozhraní vyvolávat nebo se skrze něj přihlásit k odběru dat. Rozhraní je postavené na základě signalizace, to znamená, že pro odběr dat se musí modul přihlásit. Aplikace se naopak musí přihlásit k odběru dat z modulů, aby mohla později odebraná data dále distribuovat sítí.

- **ReceiveControlData** - umožňuje modulu odebírat kontrolní a jiná data.
- **ReceiveFrameData** - umožňuje odebírat frame data, tedy obrazová data z komponenty změn.
- **ReceiveSlideData** - umožňuje odebírat slide data, tedy data obrazová motion jpeg data.
- **ControlDataEventHandler** - slouží pro odesílání kontrol dat a jiných dle preferencí vývojáře modulu.
- **FrameDataEventHandler** - slouží pro odeslání frame dat.
- **SlideDataEventHandler** - slouží pro odeslání slide dat.

IPlugin Reprezentuje samotnou instanci zásuvného modulu. Umožňuje přístup k UI rozhraní modulu a IPluginHost rozhraní pro zjištení hostitele pluginu. Povinná implementace se skládá z metod Start, SignalToStop a Close.

- **Start** - odstartuje plně modul a jeho funkcionalitu.
- **SignalToStop** - signalizuje modulu, že má ukončit své aktivity.
- **Close** - poslední signalizace oznamující, že bude modul po tomto zničen, měl by uvolnit své zdroje pomocí Dispose.

PluginMetadataAttribute Atribut specifikující metadata každého modulu. Každý modul musí tento atribut implementovat, aby byl validní pro zavedení aplikací a PluginManagerem. Metadata se skládají z ikony modulu, interního jména, jména pro zobrazení v UI aplikace a vlastnosti specifikující, zda modul zobrazí své rozhraní i v modu vyučujícího.

- **CamInfo** - definuje použitý typ kamery pro přenos obrazu, pokud modul takový přenos realizuje a definuje, aplikace pro něj vytvoří sadu objektů usnadňujících zpracování.
- **HasTeacherUI** - definuje, zda modul nabízí své UI rozhraní v módu učitele.
- **IconResource** - relativní cesta ke zdroji ikony, která je zobrazena v nabídce modulů, musí jít o obrázek nebo ikonu, který je zakompilován do dll souboru modulu ve složce /Resources jako typ "Embedded Resource"o velikosti 16x16 pixelů.
- **PluginDisplayName** - jméno modulu zobrazené v nabídce modulů aplikace.
- **PluginUniqueInternalName** - interní unikátní jméno modulu, jde o jméno dll souboru modulu bez přípony ".dll", využito PluginManager třídou a aplikací pro odlišování datových streamů jednotlivých modulů.

Plugin Jde o model zásuvného modulu reprezentující modul v aplikaci. Obsahuje set vlastností jako: SHA1 hash modulu, jeho interní jméno, odkaz na jeho reálnou instanci skrze rozhraní IPlugin a metadata modulu ve formě atributu.

PresentationBasePage Reprezentuje grafické rozhraní modulu, jde o formu UserControl grafické komponenty, ze které programátor modulu musí odvodit své UI rozhraní. Nabízí odběr akce v případě zahození UI. V budoucnu může obsahovat další užitečné atributy zajištující konzistentnost skrze UI rozhraní modulů.

4.2 Realizace jednotlivých modulů

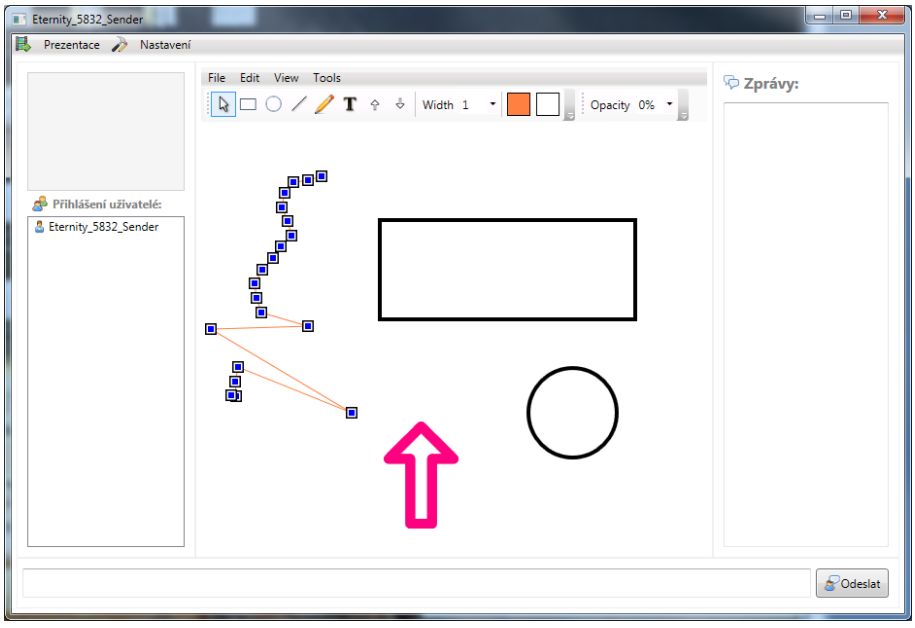
Následující kapitola se zabývá návrhem nových zásuvných modulů a analýzou hlavní funkcionality.

4.2.1 Whiteboard

Modul byl navržen pro synchronizované kreslení mezi uživateli. Uživatelé mohou vytvářet nové kreslící objekty, manipulovat s nimi a měnit jejich vlastnosti a parametry. Některé operace jako mazání jsem se rozhodl nesynchronizovat, aby si uživatelé nemazali objekty, se kterými právě pracují. Funkcionalitu modulu zajišťuje knihovna WPFCanvasTools, která je popsána v kapitole Knihovny zásuvných modulů. Kreslení probíhá v rámci canvasu poskytovaného knihovnou. Modul nabízí následující výhody.

- Využívá vlastní implementaci canvasu pro vykreslování objektů.
- Nabízí možnost synchronizovaného kreslení mezi uživateli systému.
- Data canvasu je možné uložit i lokálně a později načíst.
- Nepřímá podpora vrstev (umožňuje kreslící objekt převést do pozadí nebo popředí).
- Umožňuje pohyb, zvětšení, zmenšeni a průhlednost objektů a pozdější změny jejich vlastností.
- Vývojář modulu může poměrně jednoduše odvodit nový kreslící objekt a přidat jej do UI rozhraní modulu.

Modulu jsem navrhl následující UI rozhraní a nabídky.

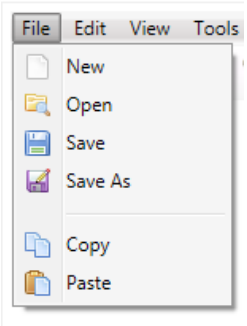


Obrázek 12: Modul whiteboard - kreslící nebo také interaktivní tabule

1. **Nabídka menu** - hlavní nabídka modulu, jednotlivé menu popíši dále v textu.

Obrázek 13: Nástrojová lišta modulu kreslení

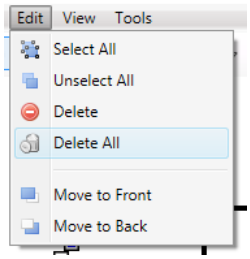
2. **Nástrojová lišta** - lišta obsahující sadu nejčastějších akcí pro manipulaci s kreslícími objekty.
   (a) **Nástroje** - jednotlivé kreslící nástroje pro canvas.
   (b) **Hrubost obrysu čáry** - hrubost čáry od 1 do 10 jednotek hrubosti.
   (c) **Barva objektu a pozadí** - dialog s výběrem barvy pro kreslící objekt a barvy pro pozadí canvasu.
   (d) **Průhlednost objektu v procentech** - od 0% do 100% průhlednosti.



Obrázek 14: Nabídka File

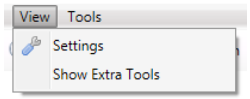
- New - nová kreslící plocha.
- Open - otevře uloženou kreslící plochu z minula.
- Save - uloží kreslící plochu do stávajícího souboru.
- Save As - uloží kreslící plochu do nového souboru dle výběru.
- Copy - kopie objektu.
- Paste - vložení kopie objektu.

- Select All - vybere všechny objekty v canvasu.
- Unselect All - opak výběru všech objektů v canvasu.
- Delete - smaže objekt v canvasu.



Obrázek 15: Nabídka Edit

- Delete All - promaže celý canvas.
- Move To Front - zobrazí kreslící objekt v popředí.
- Move To Back - zobrazí kreslící objekt v pozadí.



Obrázek 16: Nabídka View

- Settings - slouží pro budoucí rozšířené nastavení canvasu.
- Show Extra Tools - zobrazuje další nástroje kreslení, pokud jsou dostupné.

Nabídku Tools zde již uvádět nebudu, protože obsahuje stejné nástroje jako nástrojová lišta v sekci nástroje.

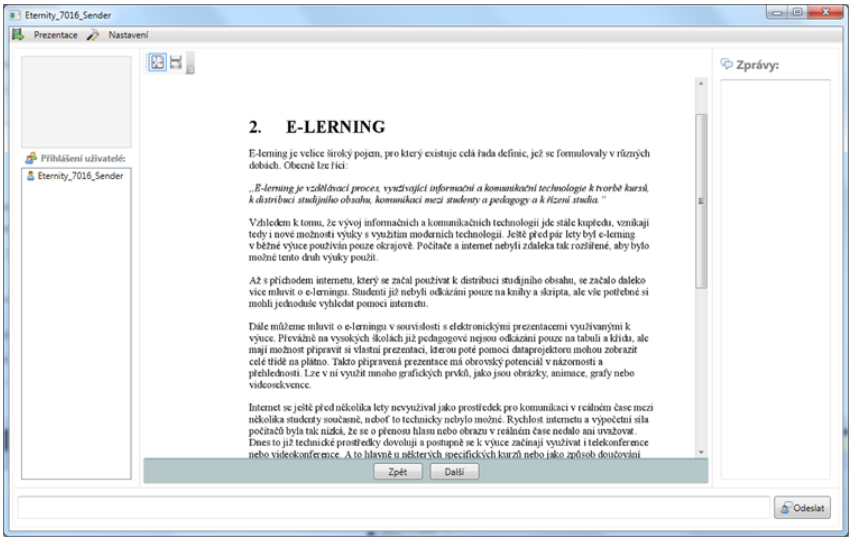
4.2.2 PowerPoint Presentation

Modul navržený pro prezentace power point souborů typu ppt a pptx. Power point prezentace jsou velmi využívané pro představení projektů nebo jiné látky. Po spuštění modulu vyučující vybere power point soubor z konkrétního umístění pro prezentaci uvnitř modulu. Poté se přesouvá mezi jednotlivými snímky v prezentaci pomocí tlačítek další a zpět. Prezentované snímky se přenášejí studentovi, který je zde v roli posluchače a nemůže prezentaci procházet.

4.2.3 PDF Presentation

Modul navržený pro prezentace pdf souborů. Pdf soubory jsou dnes velmi rozšířeným typem souborů a jsou hodně oblíbené. Může jít o přednášky, projekty, studijní materiály a mnohé další informace. Vyučující po spuštění modulu jednoduše vybere pdf dokument z daného umístění a modul jej zobrazí v prezentaci. Prezentaci tedy provádí vyučující. V modulu se dále vyučující pohybuje pomocí tlačítek další a zpět. Při každé nové stránce je studentům odeslán snímek aktuální

prezentace. Studenti jsou zde pouze v roli posluchačů, jak tomu bývá zvykem na přednáškách a jsou tak závislí na průchodu prezentace vyučujícím.



Obrázek 17: UI modulů PDF a PPTX

5 Knihovny zásuvných modulů

V následující kapitole rozeberu a popíšu jednotlivé mnou vyvinuté knihovny zajišťující kompletní většinu funkcionality každého zásuvného modulu. Každý modul je určen pro něco jiného, ale určitá funkcionalita v podobné formě může být používána skrz na skrz dalšími moduly. Také dává smysl, aby funkcionalita věnující se konkrétnímu problému nebyla natvrdo naprogramovaná přímo v konkrétním modulu, ale místo toho v konkrétní knihovně. Knihovnu s obsaženou funkcionalitou lze poté snáze použít v dalších modulech nebo v hostující aplikaci. To vychází z myšlenky znovupoužitelnosti, tedy abychom stejnou nebo podobnou funkčnost seskupili pospolu a mohli kdykoli znovu využít dále v jiných projektech. Knihovny implementované tímto způsobem tak nabízejí kompletní API pro konkrétní problematiku a snáze se tak pracuje.

5.1 MultiCapturer

Speciálně vyvinutá knihovna pro moduly sdílející obraz. Nabízí hned několik nezávislých komponent s obecným rozhraním pro zpracování obrazu. Těmi jsou:

1. **MagnifierRecorder** - zachytávač obrazu implementovaný nad MagnifierAPI technologií.
2. **DirectxRecorder** - zachytávač obrazu implementovaný nad DirectX 11.1 technologií s využitím komponenty DXGI 1.2.
3. **FrameDifferenceComparer** - vyhledávač změn v obraze.
4. **MagnifierWindowRecorder** - zachytávač obrazu okna aplikace odvozený z MagnifierRecorder třídy.

Jak bylo řečeno, komponenty mají obecné rozhraní a lze je tedy použít nezávisle jednu na druhé pro konkrétní zadanou práci. Můžeme tedy jednu komponentu využít pro zachycení snímků, které budeme ukládat do video souboru nebo jinou jen pro porovnávání obrazu z web kamery na změny. Nemusejí tedy nutně spolupracovat dohromady.

5.1.1 Implementace komponent

Hlavní funkcionalitu knihovny zajišťují první tři výše zmíněné komponenty, které jsou zastoupeny příslušnými třídami. Třídy MagnifierRecorder a DirectxRecorder jsou odvozeny z CapturerBase třídy, což je abstraktní třída definující rozhraní, které musí být povinně implementováno všemi odvozenými zachytávači obrazu. Třída CapturerBase nám tak garantuje vždy rozhraní pro volání funkcionality konkrétní nahrávací komponenty. Každý zachytávač obrazu nebo, chceme-li nahrávač, je zpravidla postaven nad jinou technologií, pomocí které obraz získává. CapturerBase nám tak umožňuje nejen ke všem zachytávačum obrazu přistupovat jednotně, ale udává i možnost vyvinout vícero zachytávacích komponent nad odlišnými technologiemi. Již dříve v kapitole

ohledně zpracování obrazu jsme zmínili, že DirectX řešení nepoběží na všech Windows platformách, konkrétně Windows Vista a Windows 7. K tomu účelu nabízí třída CapturerBase statickou metodu DetectCapturerForCurrentOS. Ta je schopna na základě parametru specifikujícího požadovaný typ nahrávací komponenty vrátit zpět podporovanou a doporučenou technologii, která bude na platformě zaručeně fungovat. Na základě vrácené technologie poté stačí použít továrnu na tvorbu nových nahrávacích komponent a předat ji zvolenou technologii. Pro příklad uvádím kód níže.

```
var chooseTech = CapturerBase.DetectCapturerForCurrentOS(CapturerType.
    DesktopCapturer);
var factoryCap = DesktopCapturerFactory.Create();
var settings = new CapturerSettings() { FPSCount = 30, ShowCursor = true,
    UseDoubleBuffering =true };
var cBase = factoryCap.GetDesktopCapturer(chooseTech, settings);
```

Výpis 7: Autodetekce a založení snímače obrazovky

Nejprve je třeba vytvořit továrnu skrze třídu DesktopCapturerFactory zavoláním bezparametrické metody Create. Ta nám vrátí nově vytvořenou továrnu pro tvorbu nahrávacích zařízení. V následujícím kroku si můžeme vytvořit pomocí třídy CapturerSettings nové nastavení pro nahrávací komponentu a hned dále zavolat nad vytvořenou továrnou metodu GetDesktopCapturer, která vyrobí nové nahrávací zařízení. Při vytváření zařízení můžeme také předat jako další parametr naše vytvořené nastavení, které zapříčiní, že nahrávací komponenta bude po vytvoření již plně nastavena. Pokud žádné nastavení nepředáme, je použito výchozí. Výchozí nastavení nemusí být nutně stejné pro každou nahrávací komponentu. Vytvořená nahrávací komponenta je vrácena jako bázová třída CapturerBase. Pokud potřebuje programátor volat nějakou funkčnost týkající se přímo komponenty, musí bázovou třídu přetypovat na konkrétní instanci objektu komponenty. V příloze na obrázku č. 22 je diagram.

5.1.2 Návrhový vzor Factory

Samotná továrna na tvorbu nahrávacích komponent samozřejmě využívá interně návrhový vzor factory. To není jediný vzor, který naše továrna využívá, dalším je double checking pattern a lazy loading. Double checking pattern je vzor používaný Microsoftem pro synchronizaci přístupu mnoha vláken. Tento vzor tedy využíváme k tomu, aby jsme zajistili, že továrna bude vytvořena jen jednou, tedy bude mít pouze jednu instanci. V normálním případě si návrhový vzor factory neporadí s tím, když je volán více vlákny v jednu chvíli a vlákna tak mohou způsobit race condition. Race condition je vzácný a náhodný jev, který může v tomto případě zapříčinit vytvoření vícera objektů, ačkoliv by měl být vytvořen jen jeden. Abychom tomuto problému předešli, využívá naše továrna tento návrhový vzor pro plnou synchronizaci. Dále využívá ještě lazy loading, který zajišťuje, že objekt není vytvořen předem, ale až ve chvíli, kdy jej chceme

vytvořit. Lazy loading tedy snižuje zátěž na paměť tím, že eliminuje tvorbu objektů, když nejsou potřeba. Její kód uvádím níže.

```
public static RealDesktopCapturerFactory Create()
{
   // Double checking pattern
   if (_factory == null)
   {
      lock (_lock)
      {
         // Lazy loading pattern
         if (_factory == null)
         _factory = RealDesktopCapturerFactory.Create();
      }
   }
   return _factory;
}
```

Výpis 8: Návrhový vzor Factory

5.1.3 MagnifierWindowRecorder

Poslední zmíněnou komponentou v seznamu byl MagnifierWindowRecorder, který je určen pro zachycení okna aplikace. Pro svou zachytávací funkcionalitu využívá funkcionalitu již existující třídy MagnifierRecorder, je z ní odvozen. Druhou důležitou třídou zajišťující funkcionalitu tohoto zachytávače je WindowTracker třída. Ta dokáže sledovat konkrétní okno aplikace v reálném čase a předávat zpět vektor pohybu a velikosti okna aplikace. Tyto dvě třídy dohromady zajišťují kompletní funkcionalitu komponenty. Instanci komponenty můžeme vytvořit přímo nebo využít továrnu zastoupen třídou AppCapturerFactory. Použití továrny je stejné jako u již výše uváděné továrny pro tvorbu klasických zachytávačů obrazu. Továrna používá i stejný zmíněný vylepšený návrhový vzor. Vizualizaci dědičné hierarchie spolu s využitou třídou WindowTracker můžeme vidět v příloze na diagramu č. 23.

5.1.4 Vyhledávač změn v obraze

Třetí nejdůležitější komponentou zajišťující funkcionalitu modulů pracujících s obrazem je vyhledávač jednotlivých změn v obraze. Vyhledávač je poměrně hodně konfigurovatelný a většina jeho vlastností je měnitelná za jeho běhu. Rozhraní jde vidět v příloze na obrázku 24.

Většina vlastností nezmění chování okamžitě ve stejnou chvíli, kdy byly vyvolány, ale až při příštím běhu komponenty. To nám zajišťuje konzistentní chování komponenty při změně jejích vlastností, důvodem je paralelně běžící kód vyhledávající změny. Pokud bychom provedli změnu

nějaké vlastnosti okamžitě a tato vlastnost by měla vliv na aktuální běh paralelního kódu, došlo by k pádu. Popis jednotlivých vlastností můžeme vidět níže.

- **AreasUsedHorizontally** - nastavuje počet horizontálních areálů pro každou vrstvu.
- **AreasUsedVertically** - nastavuje počet vrstev ve vertikální rovině.
- **ComparsionMode** - nastavuje porovnávací mód pro detekci změn v obraze.
- **JPEGEncodingQuality** - nastavuje kvalitu extrahovaných areálů při použití JPEG kodéru.
- **ThreadsUsed** - vyjadřuje počet použitých vláken pro paralelizaci výpočtů.

Totožné parametry jsou použity i v konstruktoru třídy. Vytvoření a vyvolání práce na komponentě je velmi jednoduché a intuitivní, viz. kód.

```
// Vytvor komponentu
var cmp = new FrameDifferenceComparer(2, 8, 8,
FrameCompareMode.Exact, CodecFormatType.JPEG_TURBO, 80);
// Porovnej predchozi a aktualni snimek
cmp.Compare(frame_prev, frame_cur);
```

Výpis 9: Vytvoření a nastavení komponenty

Stačí nám tedy pomocí konstruktoru vytvořit novou instanci třídy FrameDifferenceComparer pro vyhledávání změn. V konstruktoru nastavíme kolik vláken bude použito pro paralelní výpočet, specifikujeme rozdělení snímku pomocí areálů a vrstev, určíme mód přesnosti porovnání, výstupní formát extrahovaných změn a při použití JPEG formátu jeho kvalitu. Následně voláme metodu Compare, které předáme parametry třídy Frame jako předchozí a aktuální snímek pro porovnání. Metoda Compare provede ověření, zda-li jsou snímky stejné velikosti a při splnění podmínky následně zavolá paralelní algoritmus pro vyhledání změn. Zpět je nám vrácen kontejner obsahující změny jako objekt PixelAreaContainer.

5.1.5 CodecEmulator

Extrahované změny v podobě areálů nemusejí být kódovány jen do JPEG formátu, lze zvolit i další dodatečné formáty bezeztrátového typu. Interní kódování totiž zajišťuje třída CodecEmulator, která zpřístupňuje kódování i dekódování skrze GDI+ a zároveň skrze turbojpegCLI wrapper. Při použití GDI+ funkcionality můžeme použít formáty typu: JPEG, TIFF, BMP, PNG. Zatímco při použití turbojpegCLI pouze JPEG formát. Pro přenos skrze síť je jasnou volbou JPEG formát, díky své malé datové velikosti, pro tuto volbu je očividně vhodný turbojpegCLI wrapper, který provede kódování a případné dekódování mnohonásobně rychleji a

efektivněji. Naopak pokud data budeme zpracovávat lokálně, může být pro nás výhodné z nějakého důvodu zachovat kvalitu, a proto zvolíme GDI+ podporované formáty jako PNG, TIFF, BMP. Pohled na CodecEmulátor třídu můžeme v příloze č. 25.

5.1.6 Frame

Snímek neboli frame je nejčastějším objektem, který zpracováváme uvnitř celé knihovny. Využívají jej jak komponenty zachytávající obraz, tak komponenta pro vyhledávání změn. Snímek prezentuje jeden kus obrazových dat skládajících se z pixelů v nativním formátu BMP 32bpp BGRA. Takový typ snímku produkují zachytávače obrazu a stejný typ snímku zpracovává vyhledávač změn v obraze. Objekt snímku obsahuje všechny důležité vlastnosti o formátu obrazu, také nabízí metody pro efektivní vytvoření kopie snímku ve formě Bitmap objektu z NET Frameworku nebo čistého pole bytů. Stejně tak umí snímek zpřístupnit jako Bitmap objekt bez jakékoliv alokace nebo kopírování dat, tedy přímou cestou. Objekt snímku lze také resetovat do původního stavu pomocí metody Reset a vybraných parametrů, to umožňuje vyhnout se zcela alokacím tohoto typu objektu. Této výhody využívají zachytávače obrazu, které touto cestou neprovádějí žádné alokace snímků vrácených zpět uživateli. To snižuje značně overhead kolem alokací. Rozhraní v příloze č. 26.

5.2 WPFCaching

Řešení vyvinuté pro cachování dat u modulů PDF a PPTX. Mezi hlavní výhody patří plná synchronizace a rychlý běh při zpracování mnoha vlákny naráz. Knihovna je navíc plně generická pro jednotlivé typy vkládaných dat.

Nabízí se zde dva módy jak pracovat s daty:

- **OnLoad** - data jsou zavedena do paměti cache okamžitě.
- **OnDemand** - na data je předána pouze reference, do paměti jsou zavedena až při vyžádání.

Při klasickém cachování jsme zvyklí, že se data uloží ihned do paměti pro pozdější použití, to odpovídá módu OnLoad. V módu OnDemand je možné data získat až při vyžádání uživatelem. Takové řešení nejenže šetří výrazně paměť v cachi, ale snižuje i počet potřebných volání jako v případě módu OnLoad. V módu OnDemand je na data totiž uložena pouze jakási forma reference, která nám přesně ukazuje, kde lze data nalézt. Ke zmíněné referenci lze navíc přidat objekt typu IProvider, který nám sděluje jak data získat a do jakého formátu je převést.

Hlavní část funkcionality zajišťuje třída MemoryCache , která je kontejnerem držícím všechna cachovaná data. Jako položky MemoryCache třídy slouží instance CacheType třídy. CacheType třída drží buďto reálná data nebo referenci na data spolu s IProvider členem. IProvider rozhraní umožňuje vytvořit nové poskytovatele obsahu. Každý poskytovatel obsahu může sloužit pro jiná data a může plnit i formu konvertoru dat. Poskytovatel obsahu nám umožňuje získat data třeba z

disku, sítě nebo databáze. Dalším důležitým členem třídy CacheType je CacheTypePolicy třída, ta dovoluje spravovat datum platnosti a expirace objektu CacheType. Můžeme tak nastavit dobu platnosti pro objekt, kdy po jejím vypršení bude objekt zničen automaticky. Kromě doby platnosti a expirace objektu se zde ještě nabízí možnost vyhazovat objekty z cache na základě vybrané strategie:

- StrategyAdaptiveLRU - odstraňuje nejstarší objekt s nejnižší prioritou.
- StrategyLRU - odstraňuje objekt s nejnižší prioritou použití.
- StrategyPresentation - odstraňuje objekty na základě průchodu prezentace, použito v modulech PDF a PPTX.
- StrategyStandart - odstraňuje nejstarší objekty v cachi.

To nám dává globální kontrolu nad cachem a jeho položkami. Vybraná strategie je spuštěna vždy, když vkládáme nový objekt a v cachi již není místo. Nakonec uvádím třídní diagram s přehledem hlavních objektů v příloze č. 27.

5.3 WPFCanvasTools

Knihovna vyvinutá speciálně pro zásuvný modul Whiteboard. Definuje sadu nových primitivních kreslících objektů. Nabízí nový typ canvasu, do kterého je možné vykreslovat tyto nové kreslící objekty. Sada objektů je založena nad grafickými primitivy WPF Frameworku.

Nabízí grafický objekt typu vector, který umožňuje programátorovi nadefinovat na základě bodů kostru objektu a odvodit tak zcela nové objekty s novými tvary. Všechny objekty jsou serializovatelné a knihovna pro tento účel nabízí specifickou třídu, jako kontejner pro objekty v rámci serializace. Každý objekt je taktéž automaticky identifikován oproti ostatním objektům, které byly v rámci instance knihovny vytvořeny. To zaručuje, že je každý objekt unikátní a identifikovatelný. Tato funkcionalita slouží hlavně pro identifikaci při přenosu dat přes síť. Každá instance knihovny na opačném konci totiž zvolí zcela jiný základ pro generování ID. To nám prakticky garantuje, že stejný základ pro generování jednotlivých ID bude v každé instanci knihovny jiný. Je tedy vysoce nepravděpodobné, že by byl stejný.

5.3.1 Canvas a jeho použití

Canvas je hlavním objektem knihovny použitým v kreslícím modulu. Nabízí samotnou kreslící plochu, která udržuje všechny vykreslené objekty pospolu pro pozdější přístup. K vykresleným objektům canvasu můžeme přistupovat velmi jednoduše pomocí indexeru. Stačí tedy zadat požadovaný index a můžeme ihned manipulovat s konkrétním objektem. Canvas vystavuje většinu využívané funkcionality v podobě sady metod, vlastností a událostí, které voláme. Jeho metody nabízejí hlavně manipulaci s grafickými objekty, jako operace typu přidání, mazání, resetování,

vykreslení, změnu vrstvy, výběr objektu a změnu vlastností objektu. Zatímco vlastnosti canvasu zajišťují hlavně jeho prvotní nastavení a chování. To se odráží v tom, jaká bude výchozí barva objektů, hrubost linie nebo jaký bude výchozí textový font. Pokud půjdeme dále k událostem, jejichž odběr si nad canvasem můžeme zařídit, stojí za zmínku jen jediná událost, OnDrawingDone. Událost OnDrawingDone nám signalizuje akci po dokončení kreslení. Obyčejně se jedná o vytvoření a tedy vykreslení nového objektu do canvasu nebo o změnu parametrů některého existujícího kreslícího objektu v canvasu. Událost nám nejen oznámí typ použitého nástroje, ale i samotný kreslící objekt. Právě tato událost řeší hlavní část funkcionality modulu kreslící tabule, tedy získání nově vykreslených nebo změněných objektů a jejich zpracování a odeslání skrze síť pro jejich synchronizaci. Takto získané objekty na druhém konci můžeme zase velmi lehce předat instanci canvasu, jež se postará o jejich vykreslení nebo změnu atributů existujících objektů. Velkou výhodou je, že tuto funkcionalitu canvas provádí interně a programátor se tak o tyto věci nemusí starat.

5.3.2 Grafické objekty

Základem všech grafických objektů je třída GraphicsBase odvozená od třídy DrawingVisual z NET Frameworku. DrawingVisual je objekt reprezentující vektorovou grafiku vykreslenou na obrazovku. GraphicsBase využívá tento objekt a dále jej rozšiřuje specifikací nových vlastností a metod. Vytváří tak rozhraní pro další z něj odvozené grafické objekty, které již budou použitelné pro samotný canvas. Potomky GraphicsBase jsou:

- GraphicsRectangleBase
- GraphicsLine
- GraphicsPolyLine

GraphicsRectangleBase a GraphicsVectorBase jsou objekty, které rozšiřují GraphicsBase o další konkrétní vlastnosti. GraphicsRectangleBase definuje objekty obdélníkového tvaru, proto jsou jeho potomky:

- GraphicsVectorBase
- GraphicsEllipse
- GraphicsRectangle
- GraphicsText

Textový element, objekt elipsy nebo obdélníku jsou všechno objekty odvozené z obdélníkového tvaru. Zatímco přímka nebo objekt čáry jsou založené přímo nad GraphicsBase, protože obdélníkový tvar nevyužívají.Zajímavým objektem je GraphicsVektorBase postavený nad obdélníkovým tvarem. Právě z něj jsou odvozeny tvary šipky nahoru a dolů. Ty jsou definovány jako

množina bodů utvářejících celý tvar objektu. GraphicsVektorBase totiž umožňuje odvozovat nové tvary na základě jednotlivých bodů definujících tvar.

- GraphicsArrowDown
- GraphicsArrowUp

5.3.3 SerializationHelper

Třída usnadňující serializaci objektů. Převádí množinu grafických objektů na meziobjekty schopné serializace, zároveň skrývá některé interní vlastnosti původních grafických objektů před serializací. Tato třída tvoří finální funkčnost pro získání objektů z canvasu, které jsou následně serializovány pro přenos sítí v rámci modulu kreslení.

5.4 XMLSerialization

Knihovna pro práci s XML daty v textovém či binárním formátu. Nabízí serializaci a deserializaci dat skrze statickou generickou třídu XMLSerializer. Výhodou je právě statický přístup odkudkoliv v kódu, ať už se jedná o zásuvný modul nebo samotnou aplikaci. Generika dále zajišťuje silně typový přístup z vyšším výkonem. Jde o velmi jednoduše použitelnou knihovnu, která dokáže serializovat či deserializovat data z a do disku, paměti i sítě. Využíváme v rámci modulu Whiteboard pro převod grafických objektů na binární data a zpět.

5.4.1 XMLSerializer třída

Třída nabízí dvojici metod Load a Save. Metoda Load slouží k deserializaci dat zpět na objekt, zatímco metoda Save slouží k serializaci objektu na data. Dále je samozřejmě třeba specifikovat, zda chceme serializovat či deserializovat do nebo z binárního či textového formátu. To lze udělat na základě výčtového typu Format. Pokud programátor nespecifikuje použitý formát, je výchozím formátem vždy textový formát. Třídní diagram v příloze č. 28 zachycuje podobu třídy spolu s použitým výčtovým typem.

6 Zhodnocení výsledků

K nasazení nových a předělaných modulů pro sdílení ještě plně nedošlo a byly proto testovány hlavně v rámci interního provozu. U nových modulů je provoz v síti a zatížení zdrojů PC minimální, proto u nich měření výkonu postrádá smysl a nebudu jej provádět. Každopádně měření a srovnání modulů pro sdílení plochy a okna aplikace smysl rozhodně má a může přinést zajímavé výsledky.

Provedu tedy měření pro oba zmíněné moduly, nejprve pro starou a poté pro novou verzi každého modulu. Naměřené výsledky potom srovnám a představím v grafu. Měřit budeme u každého modulu počet vyprodukovaných snímků za sekundu a dále počet vyprodukovaných dat za sekundu v kilobitech. Data tedy budou prezentovat čistá data vyprodukovaná daným modulem předtím, než budou odeslána sítí. Pokud bychom chtěli znát velikost po doručení, stačilo by připočíst velikost hlavičky paketu síťového protokolu ForceB. Velikost hlavičky paketu je ale zanedbatelná.

6.1 Metodika měření

Měření probíhalo v rozlišení 2560x1440 px (2K rozlišení). U výsledků měření každého modulu je uveden graf s popisem měřených veličin. Po měřeních následuje celkové srovnání.

Výkon jsem změřil přidáním výpisu do konzole pro starší i novější modul v rámci Visual Studia, oba měřené moduly byly spuštěny pod visual studiem v Release módu.

Test proběhl na platformě Windows 7 a testovací prostředí nemělo spuštěny žádné aplikace nebo servisy a jiné procesy snižující výkon. V rámci prostředí běželo pouze Visual Studio a přehrávač videa určený pro testy.

Pro obě měřené veličiny jsme provedli celkem dva testy s použitím přehrávače videa. Pro oba testy byla spuštěna stejná sekvence videa, aby neovlivnila detekce změn v obraze. V prvním případě přehrávač překryl 50% plochy obrazovky a simuloval tak maximální zatížení při klasické práci na PC. Ve druhém případě byl přehrávač maximalizován, aby zabral celou plochu obrazovky a simuloval tak maximální zatížení.

6.2 Testovací sestava

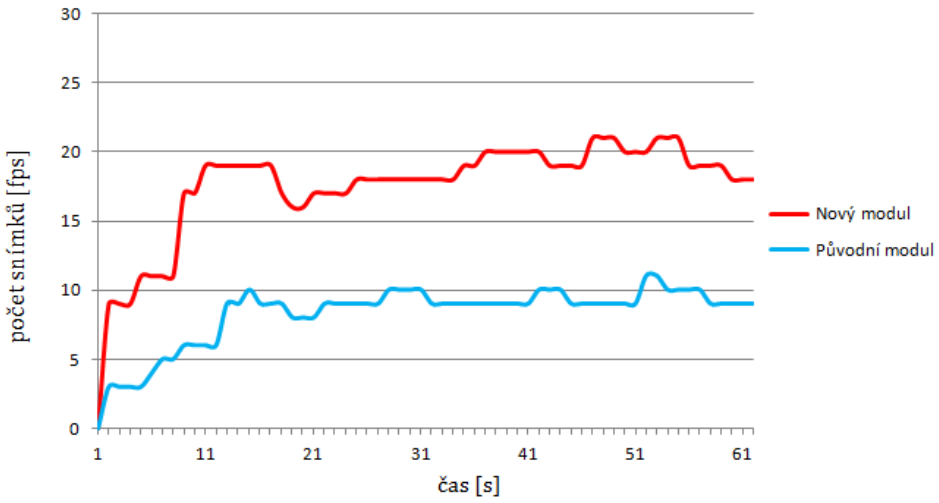
Měření bylo provedeno na následujícím HW v tabulce č. 3. Uvedeny pouze komponenty ovlivňující výkon běhu.

6.3 Měření snímků za sekundu

V tomto testu jsme změřili počet snímků pro oba moduly v každé sekundě běhu. Test běžel přesně jednu minutu a měřil změny pro 50% obrazovky, kde běžel přehrávač.

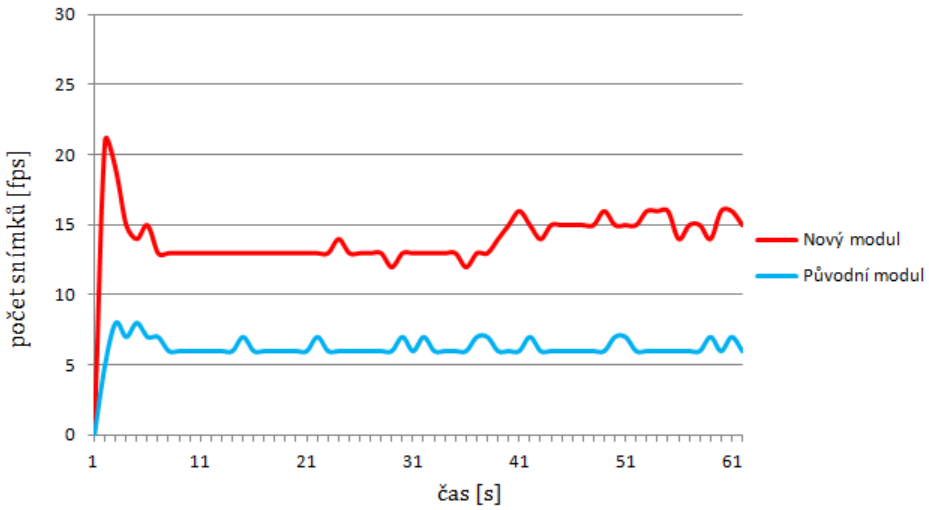
Tabulka 3: Testovací sestava a její HW ovlivňující výkon

HW komponenta	Parametry
CPU	Intel i5-4690K, 4.0 Ghz, 4jádra
GPU	Nvidia GeForce GTX 970, 1253 Mhz
RAM	Kingston 8GB, 1650 Hhz



Obrázek 18: Počet vybrodukovaných snímků za sekundu oběma moduly při 50% zatížení

V tomto testu jsme změřili počet snímků pro oba moduly v každé sekundě běhu. Test běžel přesně jednu minutu a měřil změny pro 100% obrazovky, kde běžel přehrávač.

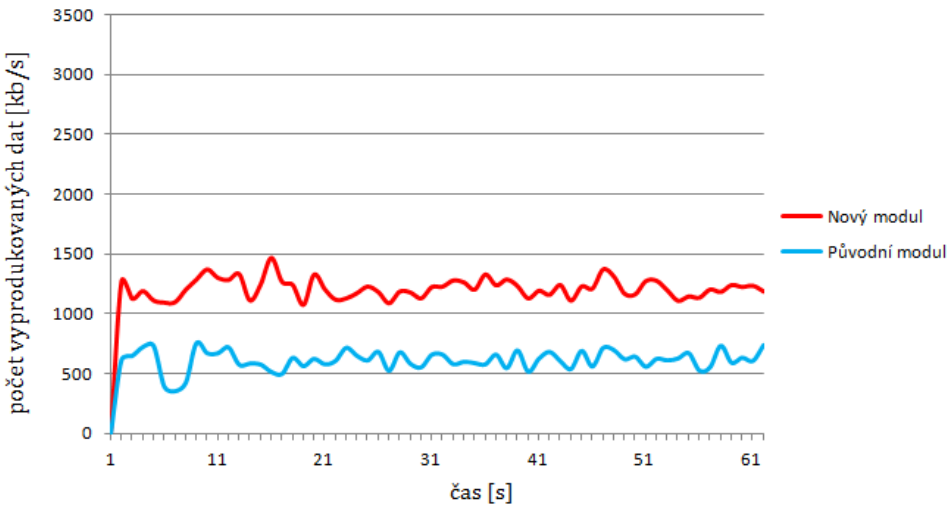


Obrázek 19: Počet vybrodukovaných snímků za sekundu oběma moduly při 100% zatížení

Množství snímků se nám viditelně u nového modulu zdvojnásobilo, ačkoliv při plném zatížení můžeme vidět, že nový modul ztratil kolem 3-4 fps oproti polovičnímu zatížení. Starý modul naproti tomu ztratil při 100% zatížení kolem 3fps, ztráta snímků je tedy podobná, ale nový modul jsi přesto udržel výrazný náskok.

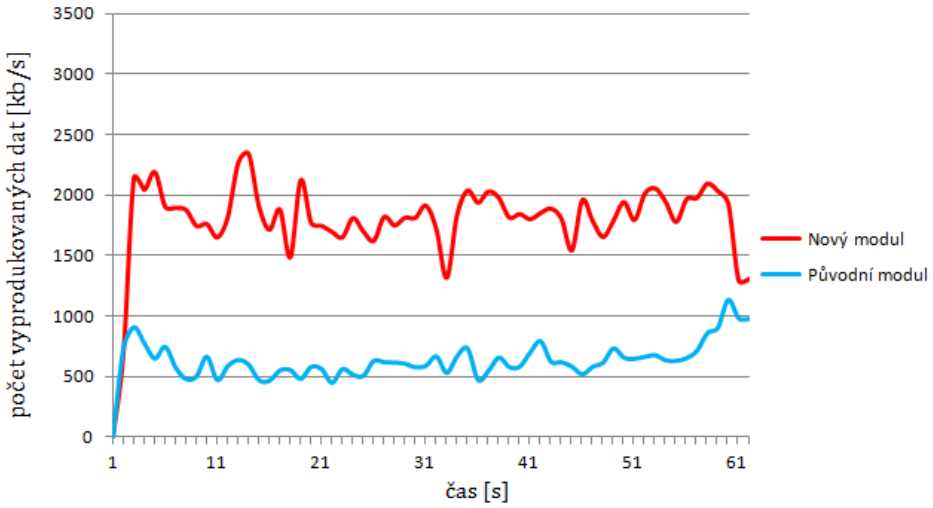
6.4 Měření datového toku

V tomto testu jsme změřili celkovou velikost vyprodukovaných dat pro každou sekundu běhu obou modulů. Test běžel přesně jednu minutu a měřil změny pro 50% obrazovky, kde běžel přehrávač.



Obrázek 20: Počet vybrodukovaných dat za sekundu oběma moduly při 50% zatížení

V tomto testu jsme změřili celkovou velikost vyprodukovaných dat pro každou sekundu běhu obou modulů. Test běžel přesně jednu minutu a měřil změny pro 100% obrazovky, kde běžel přehrávač.



Obrázek 21: Počet vybrodukovaných dat za sekundu oběma moduly při 100% zatížení

Můžeme vidět, že množství přenesených dat starým modulem se jeví nižší oproti novému, to ale není pravda, protože je třeba se podívat na počet vyprodukovaných snímků. Starý modul vyprodukoval při 50% zatížení v průměru jen 9 fps. Díky tomu taky přenesl méně dat, zatímco nový modul se blížil 20 fps a jeho datová propustnost se držela pod 1500 kb/s. Nový modul si tedy vedl lépe.

6.5 Zhodnocení

Nový modul a jeho proces zpracování obrazu je díky provedeným optimalizacím efektivnější a stabilnější. Množství produkovaných snímků je v průměru dvojnásobné oproti původnímu modulu. Počet vyprodukovaných dat novým modulem je nižší i při vyšších snímkových frekvencích. Nový modul tedy splnil svůj cíl a podařilo se mu zlepšit výkon o dvojnásobek původních hodnot.

7 Závěr

Tato bakalářská práce se zabývá problematikou zpracování a optimalizací zpracování obrazu s ohledem na zásuvné moduly pro sdílení plochy a okna aplikace, které má tímto za cíl kompletně reimplementovat a porovnat výsledky s předchozími. Dále vývojem nových zásuvných modulů dodávajících zcela nový balík funkcionality do stávajícího systému ForceB. Taktéž je jejím cílem stručně uvést čtenáře do problematiky webových seminářů a představit mu stávající systémy spolu se systémem ForceB a jeho bližším popisem. A v konečném důsledku představit i nové rozhraní pro zásuvné moduly.

Během práce nad touto bakalářskou prací jsem získal mnoho cenných zkušeností z rozdílných oblastí IT. Při řešení problémů z různých oblastí bylo třeba důkladně nastudovat a ponořit se do dané problematiky. Přes sítě, práci s obrazem, jeho formáty, až po práci s nativním kódem a optimalizacemi na platformě NET. To vše mi rozšířilo obzory, zvláště C++/CLI technologie, se kterou jsem se setkal poprvé v životě nebo injekce IL kódu. Nejdéle mi ale trvalo implementovat a otestovat knihovnu MultiCapturer a ujistit se, že je plně funkční a připravená k nasazení. Samozřejmě jsem během vývoje musel často refaktorovat kód, ať už kvůli novým nápadům nebo modifikacím kódu. Celkově se tak podařilo vytvořit efektivní řešení a úspěšně jej implementovat spolu s další zadanou funkcionalitou.

Velkou zkušeností pro mne bylo zúčastnit se workshopu ORGANON, kde jsem měl možnost zjistit více o systému ForceB a dalších zajímavých projektech. Také jsem měl možnost představit svou bakalářskou práci na workshopu ohledně systému E-logika pod názvem Data a informace.

Pokud jde o budoucí rozšíření či směr, kterým by se měl vývoj ubírat, bylo by dle mě zajímavé dále rozšířit řešení ohledně sdílení obrazu v několika směrech. Navrhoval bych rozšířit aktuální algoritmus o dokonalejší kompresi dat založenou odkazováním se na předchozí snímky, kdy je přenesen pouze snímek s referencemi na jiné snímky obsahující stejná data na daných vektorech. Taková modifikace by mohla výrazně snížit množství dat na síťovém rozhraní. Více může napovědět práce kodeků typu H.264 a H.265 a termíny jako je B-frame a P-frame. Při práci s obrazem mě také napadá možnost jej zaznamenat do souboru jako video záznam, vzhledem k tomu jak obraz přenášíme by bylo třeba vyvinout speciální typ datového kontejneru. Také by bylo zajímavé implementovat algoritmus pro detekci změn nad CUDA technologií a srovnat toto řešení s mým původním. To by pravděpodobně vedlo k dalšímu výraznému zvýšení rychlosti a výrazně nižšímu zatížení procesoru, protože bychom výpočty přesunuly kompletně na grafický adaptér. V neposlední řadě by se daly nahradit aktuální zvukové kodeky pro přenos audia pokročilejšími. A samozřejmě by bylo zajímavé vidět nové zásuvné moduly.

Literatura

[1] [1] PROKEŠ, Martin a Radoslav FASUGA. ICETA 2012 proceedings 10th IEEE International Conference on Emerging eLearning Technologies and Applications: November 8-9, 2012, Stará Lesná, The High Tatras, Slovakia. Piscataway, N.J: IEEE, 2012, Tool for desktop sharing and remote teaching - For. ISBN 9781467351201.

[2] GLYNN Jay, Watson Karli, SKINNER Morgan, ROBINSON Simon, NAGEL Christian, ALLEN K.Scoot, CORNES Ollie, GREENVOSS Zach, HARVEY Burton. C# Programujeme profesionálně. ISBN 978-80-251-2401-7

[3] R. Fasuga, P. Kašpar and M. Šurkovský, "Utilization of an Image Specific Areas in Searching,"Digital Society, 2010. ICDS '10. Fourth International Conference on, St. Maarten, 2010, pp. 279-284. doi: 10.1109/ICDS.2010.50

[4] P. Blenkhorn, D. G. Evans and A. Baude, "Full-screen magnification for windows using DirectX overlays,"in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 10, no. 4, pp. 225-231, Dec. 2002. doi: 10.1109/TNSRE.2002.806835

[5] King-Sun Fu and A. Rosenfeld, "Pattern Recognition and Image Processing,"in IEEE Transactions on Computers, vol. C-25, no. 12, pp. 1336-1346, Dec. 1976. doi: 10.1109/TC.1976.1674602

[6] Adobe Connect [online]. 07-14-2009 [cit. 2016-04-28]. Dostupne z WWW: http://www.adobe.com/products/adobeconnect.html

[7] Webex [online]. c2011 [cit. 2016-04-28]. Dostupne z WWW: http://www.webex.com/.

[8] In: Skype Web [online]. Microsoft, 2016 [cit. 2016-04-28]. Dostupné z: https://www.skype.com/cs/

[9] TIŠNOVSKÝ, Pavel. In: Root [online]. 2006 [cit. 2016-04-28]. Dostupné z: http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/#k08

[10] Microsoft Office [online]. c2011 [cit. 2016-04-28]. Microsoft PowerPoint. Dostupne z WWW: http://office.microsoft.com/en-us/powerpoint/.

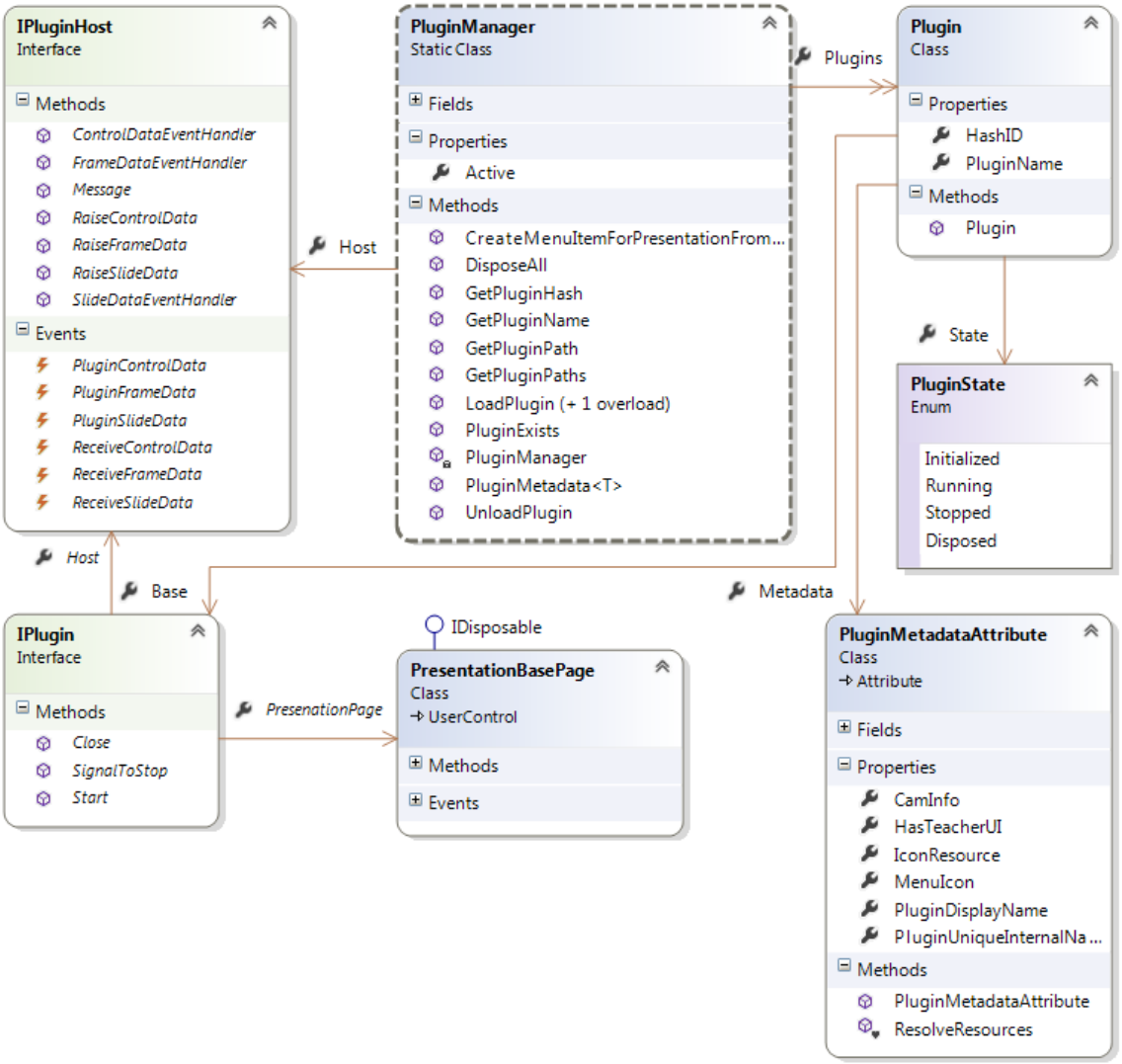
[11] Microsoft Visual Studio [online]. c2013 [cit. 2016-04-28]. Dostupné z WWW: http://www.microsoft.com/cze/msdn/vstudio/

[12] Windows Dev Center [online]. 2016 [cit. 2016-04-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466%28v=vs.85%29.aspx

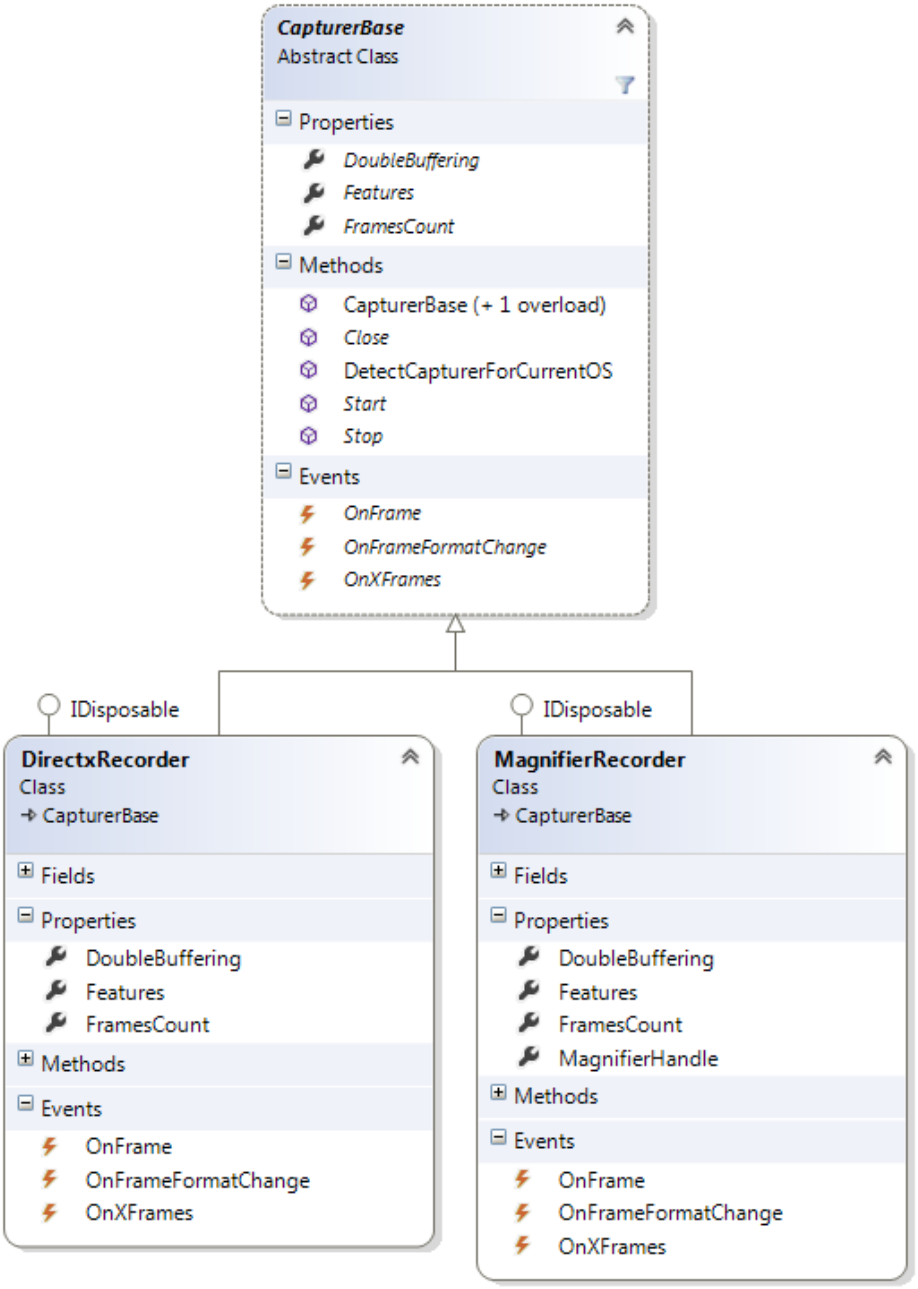
A Přílohy

Na přiloženém CD se nacházejí tyto adresáře:

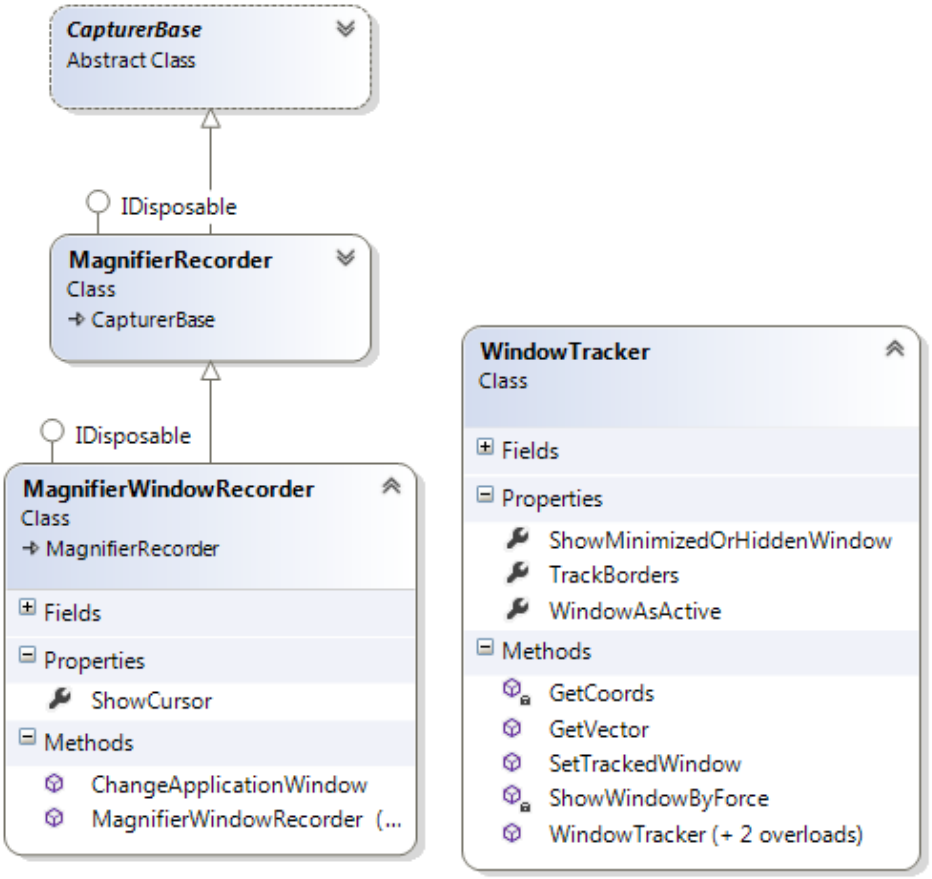
- /text Bakalářská práce - text práce ve formátu pdf
- /code Zdrojové kódy celého řešení



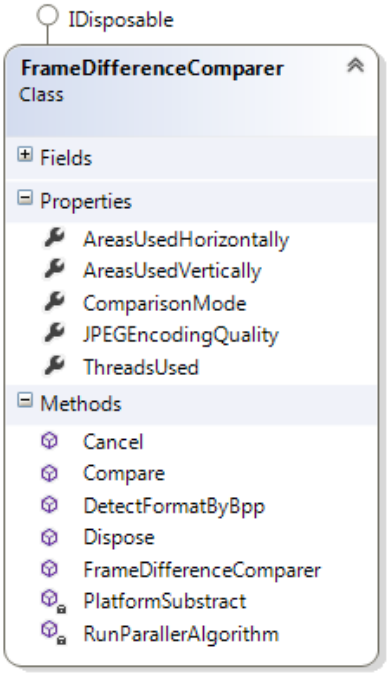
Obrázek 22: Třídní diagram zachycující nové rozhraní pro správu zásuvných modulů



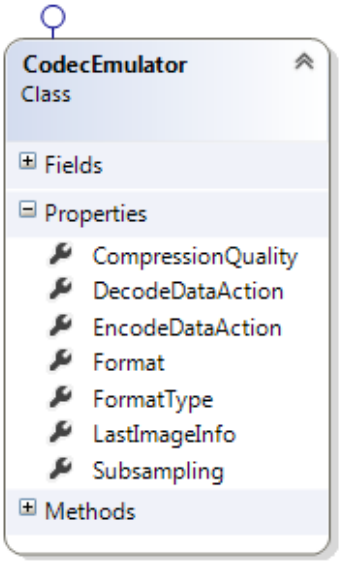
Obrázek 23: Rozhraní bázové třídy CapturerBase a jí odvozených tříd



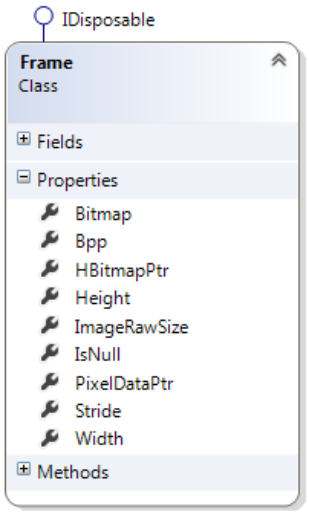
Obrázek 24: WindowTracker a hierarchie dědičnosti



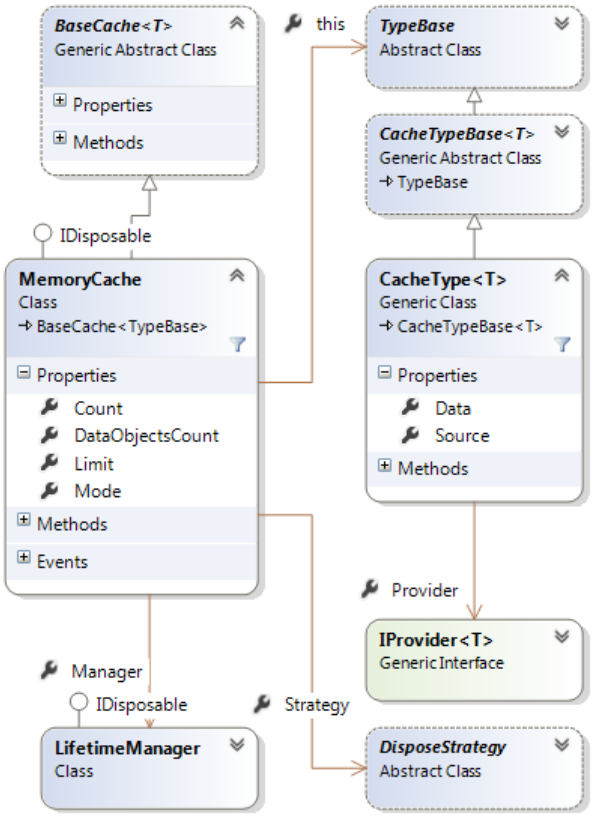
Obrázek 25: Diagram zobrazující komponentu pro vyhledávání změn v obraze



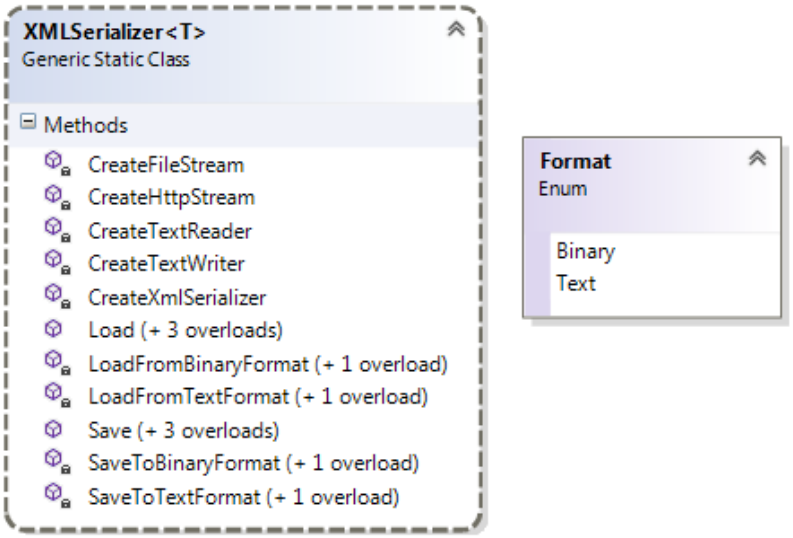
Obrázek 26: CodecEmulator umožňující emulovat kódování a dekódování skrze GDI+ a turbojpegCLI wrapper



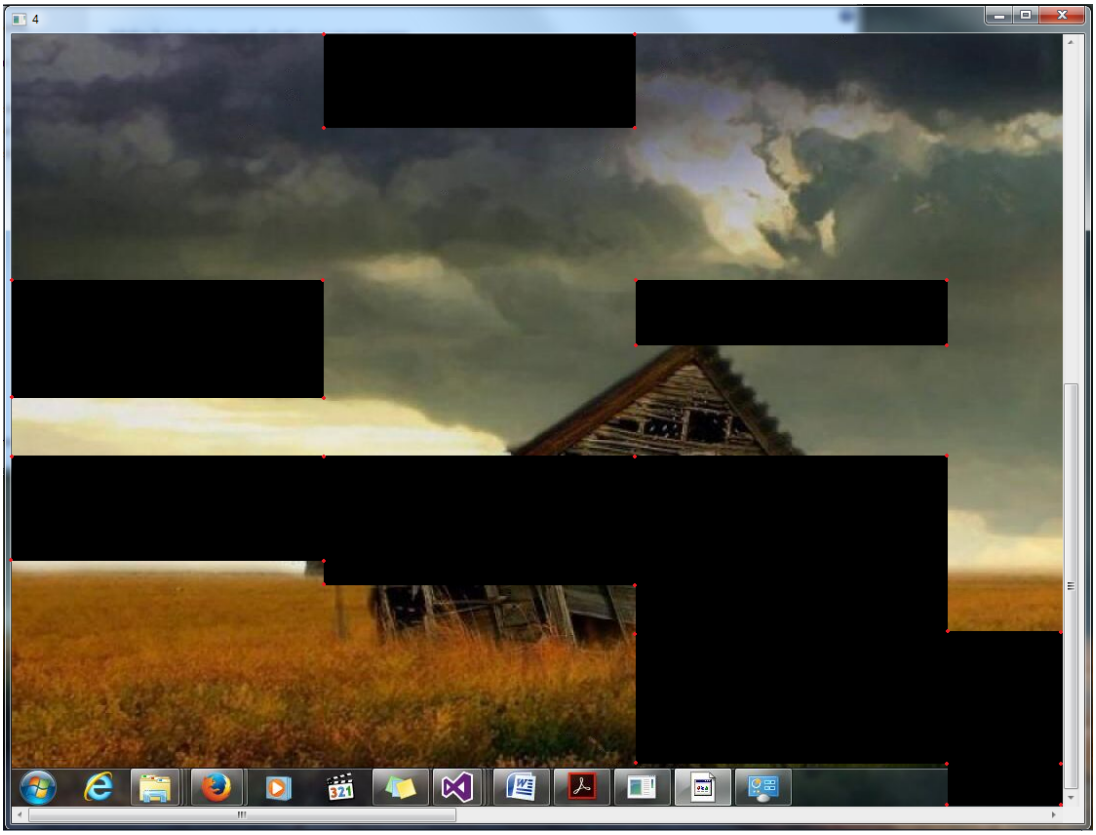
Obrázek 27: Vlastnosti snímku vyprodukovaného zachytávací komponentou nebo vytvořeného uměle



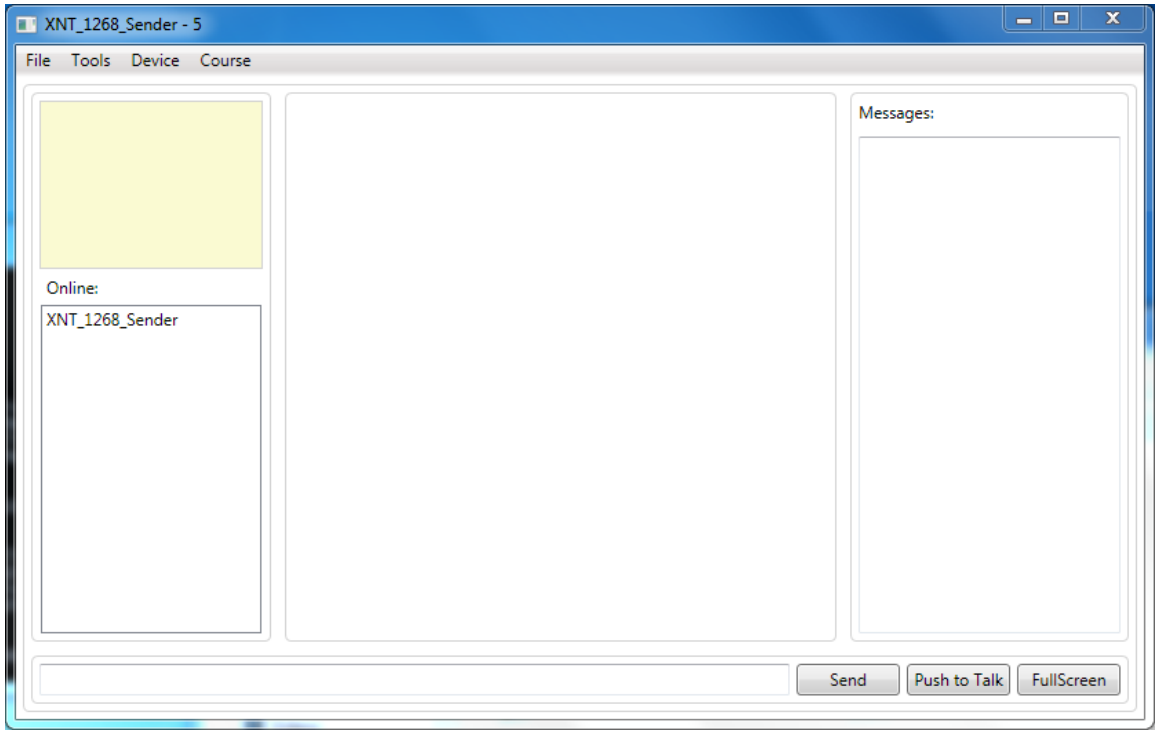
Obrázek 28: Třídní diagram hlavní části cachovací knihovny



Obrázek 29: Diagram třídy XMLSerializer a použitý výčtový typ



Obrázek 30: Zobrazení nezměněných (černých) a změněných areálů (ostatní) v obraze.



Obrázek 31: Původní grafické rozhraní klienta