VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

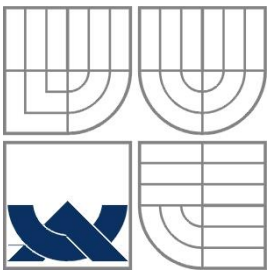
SYSTÉM PRO SDÍLENÍ SKENERŮ PO SÍTI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

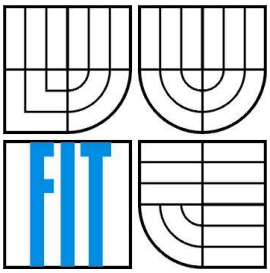
AUTOR PRÁCE
AUTHOR

Bc. MARTIN RICHTER

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SDÍLENÍ SKENERŮ PO SÍTI

SYSTEM FOR SHARING SCANNERS OVER NETWORK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN RICHTER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2015

Abstrakt

Cílem této diplomové práce je vytvoření systému pro sdílení skenerů po počítačové síti. Systém je navržen pro rozhraní TWAIN a WIA na operačním systému Microsoft Windows a pro rozhraní SANE na linuxových systémech. Implementace proběhla pomocí programovacího jazyka C++, knihoven Boost a frameworku Qt. Vytvořené pomocné knihovny mají význam i mimo tuto práci, především framework TWAIN++. Výsledný systém umožňuje sdílení skenerů po síti a připojení k nim pomocí libovolného skenovacího rozhraní.

Abstract

The purpose of this master's thesis is creation of a system capable of sharing scanners over computer network. The target scanner interfaces are TWAIN and WIA on Microsoft Windows operating system, and SANE on GNU/Linux. C++ programming language, Boost libraries and Qt framework were used to implement the programming part of this work. Several smaller helper libraries were implemented that are useful even outside this work, most notably TWAIN++ framework. The resulting system enables the user to share scanners over network, and scan using any of the aforementioned interfaces.

Klíčová slova

skener, scanner, TWAIN, WIA, SANE, backend, frontend, miniovladač, mikroovladač, datový zdroj, síť, TCP/IP, klient, server

Keywords

scanner, TWAIN, WIA, SANE, backend, frontend, minidriver, microdriver, data source, network, TCP/IP, client, server

Citace

Richter Martin: Systém pro sdílení skenerů po síti, diplomová práce, Brno, FIT VUT v Brně, 2015

Systém pro sdílení skenerů po síti

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Pavla Očenáška, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Richter
25. 5. 2015

Poděkování

Děkuji svému vedoucímu Ing. Pavlu Očenáškovi, Ph.D. za odbornou pomoc při řešení této práce.

© Martin Richter, 2015
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

V současné době jsou domácí i firemní počítačové sítě tvořeny zařízeními od mnoha výrobců, která jsou obsluhována různými operační systémy. Softwarové produkty musí tedy komunikovat pomocí jednotných rozhraní a protokolů nezávislých na použité platformě. V případě skenerů však existuje těchto rozhraní více. Omezíme se pouze na stolní počítače s operačními systémy Windows a Linux, kde je ke komunikaci se skenery použito těchto aplikačních rozhraní: TWAIN [1], WIA [2] (obojí Windows) a SANE [3] (Linux). Tato tři rozhraní nejsou kompatibilní, aplikace, která používá jedno z nich, nemůže použít skener připojený k jinému. Cílem tohoto projektu je navrhnout a implementovat systém, který umožní aplikaci využívající libovolné ze tří rozhraní použít skener zpřístupněný pomocí jakéhokoliv jiného nebo i stejného rozhraní. Díky využití počítačové sítě bude navíc možné skenovat na skeneru připojenému ke vzdálenému zařízení.

V kapitole 2 jsou popsány technologie, které jsou užity při implementaci systému. Jde o jazyk C++, kolekci síťových protokolů TCP/IP, knihovnu Boost a Framework Qt.

V následující kapitole 3 jsou představena skenovací rozhraní, pro operační systémy Linux jde o rozhraní Scanner Access Made Easy (SANE), na systém Windows cílí rozhraní TWAIN a Windows Image Acquisition (WIA).

Kapitola 4 se již zabývá návrhem systému pro sdílení skenerů po síti. Jsou zde navrhnuti síťoví klienti pro Linux a Windows a služba serveru pro Windows. Taktéž je tu specifikován síťový protokol pro přenos příkazů TWAIN.

Implementace systému je popsána v kapitole 5. Zmiňuji tu i množství jednodušších podpůrných knihoven. Cílem nově vytvořených frameworků SANE++ a TWAIN++ je podstatné zjednodušení vytváření všech částí architektur odpovídajících rozhraní. Nad těmito frameworky operují knihovny, které implementují klientskou stranu spojení síťového protokolu SANE a obě strany nového síťového protokolu pro přenos operací rozhraní TWAIN.

Testování proběhlo na dvou fyzických skenerech (resp. multifunkčních zařízeních) a virtuálních zařízeních. V kapitole 6 jsou zapsány jednotlivé výsledky pro každé rozhraní a zařízení a provedl jsem jejich celkové vyhodnocení.

Způsob návrhu a implementace systému mohou mít za následek omezení funkčnosti či nepodporu některých částí skenovacích rozhraní. Případy týkající se tohoto projektu jsem podle jednotlivých rozhraní zanesl do kapitoly 7.

Kapitola 8 obsahuje náměty na další a úpravy rozšíření projektu. Vycházím zde z výsledků testů a seznamu omezení tak, aby případná pokračování nebyla zatížena zjištěnými chybami a omezeními.

2 Použité technologie

2.1 Jazyk C++

C++ je kompilovaný programovací jazyk podporující několik programovacích paradigmat, především procedurální, objektově orientované a generické. Jako implementační jazyk byla zvolena verze označovaná jako C++11 [4] [5], neboť se jedná o současnou nejnovější verzi se stabilní podporou v použitých překladačích.

2.2 TCP/IP

TCP/IP slouží jako označení rodiny internetových protokolů sestávající ze síťového protokolu IP (verze 4 [6] a verze 6 [7]; dále jako IPv4 a IPv6) a transportních protokolů TCP [8] a UDP [9]. Ostatní protokoly nejsou pro tuto práci podstatné. Jedná se o model čtyř nezávislých vrstev: linková, internetová, transportní a aplikační.

Linková vrstva popisuje spojení mezi dvěma sousedními stanicemi, v této práci se jí nebudeme zabývat. IP je protokol internetové vrstvy, ustavuje spojení mezi dvěma body. Pro naše účely je nutné podotknout, že IPv4 používá 32bitové a IPv6 128bitové adresy. Nad IP operují protokoly TCP a UDP. TCP zajišťuje spolehlivý přenos informace v souvislém toku. UDP tyto záruky neposkytuje, pouze zasílá datagramy. Oba protokoly definují 16bitové číslo portu, které je využito k identifikaci aplikací na obou stranách spojení. V nejvyšší aplikační vrstvě modelu je stanoven protokol přenosu dat a informací podstatných pro samotné aplikace.

2.3 Boost

Boost [10] je uskupení volně dostupných knihoven pro jazyk C++. Knihovny zde obsažené jsou až na výjimky multiplatformní. Dále jejich použití nevyžaduje inkluzi celého balíku, nýbrž jen nutných závislostí. Toto spolu s faktem, že knihovny jsou z podstatné části tvořeny jen hlavičkovými soubory, vede na celkově menší využití místa. Díky permisivní licenci [11] není statický překlad problém ani u komerčních projektů. Lze tak jednoduše snížit počet závislostí a předejít tzv. „DLL peklu[1]“ („DLL hell“). Statická kompilace také umožňuje lepší celkovou optimalizaci.

Z význačných knihoven jmenujme `Boost.Asio` [12], `Boost.Lexical_Cast` [13] a `Boost.PropertyTree` [14]. `Boost.Asio` poskytuje generické třídy a funkce pro nízkoúrovňový přístup k vstupním a výstupným operacím. Nad tímto základem je vystavěn přístup k síti typu TCP/IP.

[1] Označení množiny problémů při použití knihoven od nekompatibility verzí až po obrovské stromy závislostí.

Ve spojení s knihovnou OpenSSL [15] lze vytvářet i šifrovaná síťová spojení. Také je zde přítomno rozhraní pro časovače (timers), sériové porty a zpracování signálů operačního systému.

`Boost.Lexcal_Cast` je jednoduchý, ale mocný nástroj pro konverzi typů na řetězce a zpět za běhu programu. Oproti funkcím standardní knihovny poskytuje jednotné rozhraní pro všechny konverze (podobné konverzním operacím jazyka C++). Chyby (např. špatný vstup, přetečení nebo podtečení rozsahu cílového typu) jsou indikovány výjimkou.

Třetí zmíněná knihovna, `Boost.PropertyTree`, umožňuje vytváření a práci se stromy vlastností. Jsou poskytnuty nástroje k uložení a načtení stromů ze čtyř různých formátů: XML, JSON, INI a INFO.

2.4 Framework Qt

Framework Qt [16] poskytuje kompletní nástroje pro vývoj aplikací napsaných v jazyce C++. Qt je děleno do modulů, které poskytují široké spektrum funkcionality od základních struktur, jako jsou řetězce a kontejnery, přes podporu skriptování v jazyce JavaScript, až po vytváření grafických rozhraní a modulu kompletního webového prohlížeče.

Výstavba uživatelských rozhraní (dále GUI) je možná za pomoci MOC[2] v jazyce C++ použitím tzv. widgetů. Jedná se o jednotlivé grafické komponenty, jako jsou okna, popisky, textová pole aj. Druhý způsob využívá jazyka QML[3], který popisuje GUI deklarativně (jazyk podobný JavaScriptu).

[2] Meta-Object Compiler, překladač rozšíření jazyka C++ pro potřeby Qt (např. signály a sloty, vlastnosti aj.). Více na http://doc.qt.io/qt-5/metaobjects.html.
[3] Qt Meta-Object Language, deklarativní jazyk podobný JavaScriptu pro vytváření rychlých a dynamických GUI. Více na http://doc.qt.io/qt-5/qtqml-index.html.

3 Skenovací rozhraní

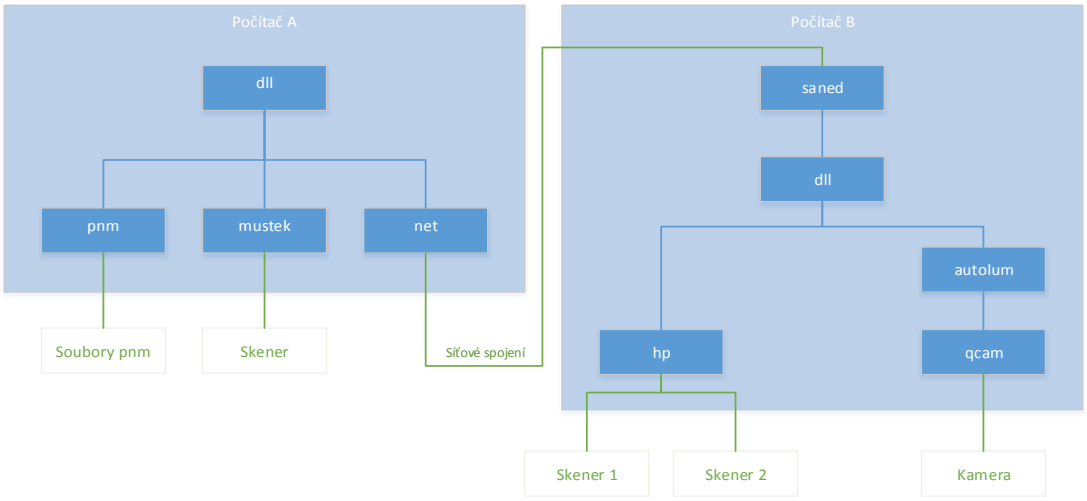
3.1 Scanner Access Now Easy

SANE [3] je multiplatformní rozhraní pro přístup ke skenerům. I přes svou multiplatformnost je používáno prakticky jen na unixových, resp. linuxových systémech. Architektura sestává ze dvou částí, tzv. frontendu a backendu.

Jako frontend je označena aplikace, která pomocí SANE přistupuje k připojeným zařízením. Jedná se obvykle o příjemce obrazových dat, tedy aplikaci s grafickým uživatelským rozhraním nebo aplikaci v příkazovém řádku.

Protipólem je backend. Ten implementuje ovladač konkrétního zařízení nebo skupiny zařízení a pomocí SANE je zpřístupňuje frontendům. Speciálním případem je tzv. meta backend, který nekomunikuje přímo se zařízením, ale zprostředkovává nebo nějakým způsobem ovlivňuje komunikaci s jiným (meta) backendem. Vůči aplikaci se tedy chová jako backend a vůči zpřístupňovanému backendu jde o frontend.

Na obrázku 1 můžeme nalézt příklad hierarchie SANE. Na počítači A se aplikace připojují k backendu `dll`, který jim zprostředkovává ostatní backendy `pnm`, `mustek` a `net`. První dva v pořadí zpřístupňují databázi obrázků a skener, jedná se tedy o čisté backendy. Třetí je meta backend, neboť pomocí sítě a služby `saned` (frontend) na počítači B zpřístupňuje zařízení tohoto vzdáleného stroje. Backend `hp` zde připojuje dvě různá zařízení, skenery 1 a 2. Ještě si všimněme položky `autolum`, tato automaticky upravuje jas obrazu získaného z kamery připojené pomocí `qcam`. Jde o demonstraci faktu, že meta backendy mohou s daty jimi procházejícími libovolně manipulovat, ne jen přeposílat dále jako v případě `net`.

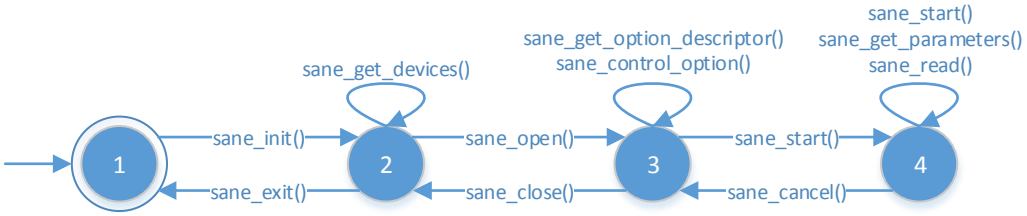


Obrázek 1: Příklad hierarchie SANE (podle [3])

Oficiální distribuce SANE obsahuje množství backendů (ovladačů) pro různá zařízení a meta backendy s různou funkcionalitou včetně `dll` a `net`. Aplikace často komunikují přímo jen s `dll` (obvykle sdílený objekt, forma dynamické knihovny), jež dle nastavení zpřístupní zařízení ostatních backendů. Není tedy nutné provádět jejich detekci ve vlastní režii.

Obrazové body jsou přenášeny v rámcích a to po řádcích zleva doprava a odshora dolů. Každý řádek obsahuje stejný počet obrazových bodů jako řádky ostatní, přičemž tyto mohou končit neurčeným počtem neobrazových bajtů („vycpávka“ neboli padding, obvykle nulové bajty). Rámec může představovat kompletní obrázek v odstínech šedi nebo v barvě (RGB – červená, zelená a modrá barva), méně častou možností jsou rámce sestávající jen z jediné barvy RGB, které jsou přeneseny zvlášť a poté složeny do jediného barevného obrázku. V případě barevného rámce jsou jednotlivé barvy přenášeny prokládaně v pořadí červená, zelená, modrá. U jednobitové barevné hloubky jsou složky prokládány po bajtech, tedy se nejprve přenese osm červených složek osmi sousedních obrazových bodů, poté stejným způsobem jejich zelené a konečně modré barvy.

Obrázek 2 ukazuje workflow aplikačního rozhraní SANE. Jedná se o jednoduché a navíc intuitivní řešení. Po načtení backendu, které je definované jeho formou a použitým operačním systémem, se spojení nachází ve stavu 1, hned poté je zavolána funkce `sane_init()`. Zde je provedena inicializace backendu v závislosti na jeho implementaci a spojení je povýšeno do stavu 2. V tuto chvíli je možné získat seznam všech dostupných zařízení pomocí `sane_get_devices()` a jedno z těchto zařízení vybrat (lze přeskočit a vybrat zařízení takto získané v minulosti) k otevření funkcí `sane_open()`, kdy je proveden přechod do stavu 3. Zavoláním `sane_exit()` se lze oproti tomu vrátit do stavu 1 a celý proces zopakovat nebo spojení ukončit (např. vypnutí aplikace). Ve stavu 3 je provedeno nastavení backendu. Funkce `sane_option_descriptor()` získává popisnou strukturu jedné položky. Každé položce je přiděleno celé číslo od nuly výše, přičemž mezi čísly nejsou mezery. Položka číslo nula je jediná, jejíž přítomnost je zaručena rozhraním; popisuje celkový počet položek nastavení otevřeného zařízení. Samotné čtení a změna nastavení těchto položek jsou prováděny pomocí `sane_control_option()`. Zavoláním `sane_close()` je sezení vráceno do stavu 3, `sane_start()` pak zahájí získávání obrazových dat (rámce) v stavu 4. Zde je možné pomocí `sane_get_paraneters()` získat informace o současně přenášeném rámci (např. typ a velikost), `sane_read()` je užito k čtení dat do paměti. Pro ukončení čtení a návrat do stavu 3 se využije `sane_cancel()`. Ve stavu 4 je však možno volat i `sane_start()`, který byl před tím využit pro přesun do tohoto stavu. Zde indikuje začátek čtení dalšího rámce, je-li předpokládán (např. v případě přenosu barevných složek v oddělených rámcích).



Obrázek 2: Workflow SANE (volně podle [3])

3.1.1 Položky nastavení

Množství definovaných položek nastavení a jejich možnosti ovlivňují použitelnost aplikačního rozhraní. V případě SANE jde o dobře nastavený minimalistický systém, který však upřednostňuje uživatele, programová volba nastavení je velmi omezená, většinou nemožná.

Každá položka nese jméno, řetězec pro účely její identifikace, který se obvykle nezobrazuje uživateli, a pro potřeby zobrazení uživateli i titulek a popis. SANE však definuje pouze sedm standardních položek („well-known“, resp. 3+4), veškeré další položky (především název a význam) jsou vymezeny výrobcem backendu. Díky přítomnosti titulku a popisu jsou tyto dodatečné volby snadno nastavitelné uživatelem, avšak nelze je jednoduše vyhodnocovat automaticky. Standardní položky jsou tyto:

- celkový počet položek (jediná vždy přítomná volba),
- rozlišení v obrazových bodech na palec,
- režim náhledu,
- výseč obrazu pro skenování – horní levý a dolní pravý okraj (celkem čtyři souřadnice dvou různých bodů).

3.1.2 Síťový protokol

Díky použití jednoduchých struktur s přesně danými velikostmi a absenci kódu závislého na platformě je SANE plně síťově transparentní. Specifikace SANE definuje síťový aplikační protokol, přičemž je vyžadován spolehlivý transportní protokol. Výchozí implementace používá TCP/IP.

U každé z operací zmíněných v kapitole 3.1 je uveden formát dotazu i odpovědi. Backend `net` a přidružená služba `saned` však tento protokol mírně modifikují. Jiné implementace dodržující přesně protokol podle [3] a využívající síť TCP/IP tedy nejsou kompatibilní s oficiální distribucí. Analýzou zdrojového kódu služby i backendu byly nalezeny tyto rozdíly a nepřesnosti, které je nutné vzít v úvahu při vývoji kompatibilní aplikace:

- Při automatickém nastavení zasílá operace `sane_control_option()` serveru hodnotu a její typ jen při revizi protokolu dvě a nižší, novější revize protokolu tyto položky neposílají (v případě čtení a modifikace jsou obě položky posílány vždy).
- Záznam s obrazovými daty obsahuje po speciální délce `0xFFFFFFFF`, která indikuje ukončení přenosu, také jeden bajt se stavem odpovídajícím výsledku posledního volání `sane_read()`. V definici protokolu zmínka o stavovém bajtu chybí, navíc je zde nekonzistence s ostatními operacemi, kde je stav přenášen na čtyřech bajtech.

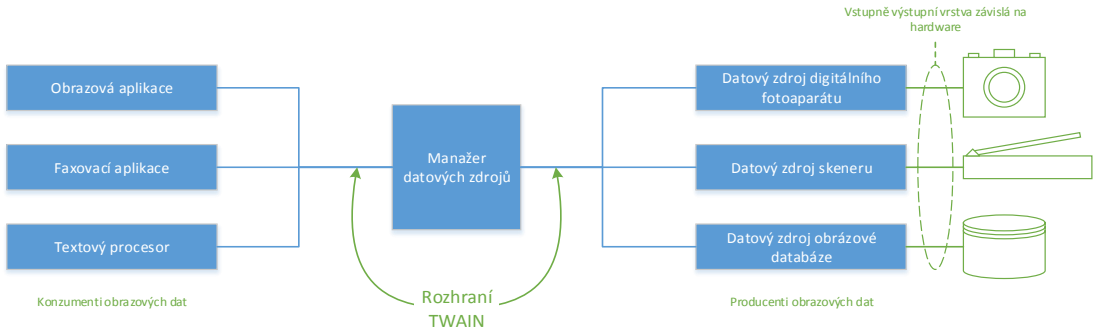
3.2 TWAIN

TWAIN specifikovaný v [1] je multiplatformní aplikační rozhraní pro nastavení a přenos dat z obrazových a zvukových zařízení. Rozhraní původně vzniklo pro operační systémy Microsoft Windows. Podpora pro Mac OS X a Linuxu byla přidána až několik let poté, kdy již na těchto platformách dominovala rozhraní jiná, proto se zde omezíme jen na software společnosti Microsoft. Přenos zvuku není v případě skenerů podstatný, nebude tedy dále popsán.

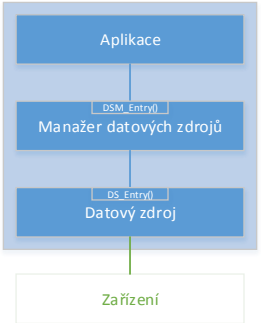
Architektura rozhraní (ilustrace na obrázku 3) sestává ze tří klíčových komponent: aplikací, manažera datových zdrojů („data source manager“, dále jen DSM) a datových zdrojů („data source“, dále jen DS). DSM funguje jako rozhraní mezi aplikacemi a dostupnými DS. Jedná se o sdílenou knihovnu (sdílený objekt) poskytovanou sdružením TWAIN Working Group. Spravuje jednotlivé DS, předává jejich seznam aplikaci a moderuje spojení mezi aplikací a DS. Pro tyto účely poskytuje jedinou univerzální operaci `DSM_Entry()`.

Jako aplikace je v kontextu TWAIN označen jakýkoliv software, který je připojen k DSM a jedná se o koncového konzumenta dat. Obvykle jde o program zpracovávající obrázky. Aplikace nastavuje veškeré parametry spojení.

Datový zdroj DS funguje jako most mezi TWAIN a konkrétním zařízením. Jedná se tedy ve většině případů o součást závislou na platformě, neboť zde probíhá komunikace s připojenými zařízeními (pokud nejde např. o databázi obrázků). Každý DS poskytuje GUI, které může aplikace využít. GUI je specifické pro připojené zařízení, a proto umožňuje jeho lepší kontrolu. Vstupním bodem DS je operace `DS_Entry()`. Komunikace aplikace, DSM a DS je ukázána na obrázku 4.



Obrázek 3: Architektura TWAIN (podle [1])



Obrázek 4: Komunikace v TWAIN (podle [1])

Aplikace komunikuje s DSM a DS (přes DSM) pomocí DSM_Entry(), DS komunikuje s aplikací a DSM stejným způsobem, popř. pomocí funkce zpětného volání (callback). DS_Entry() pak slouží ke komunikaci DSM s DS (směr od DSM). Význam volání těchto funkcí specifikují tzv. triplety. Jde o trojice parametrů, které určují kategorii operace (konstanty s předponou DG_), typ datového argumentu (předpona DAT_) a akci k provedení (MSG_). Současný standard definuje kategorie DG_CONTROL pro operace určitým způsobem ovlivňující otevřený komunikační kanál, DG_IMAGE, který indikuje práci s obrazovými daty, a DG_AUDIO, což je obdoba pro zvuková data. Typ argumentu, DAT_, slouží k identifikaci typu datové struktury, jež byla předána jako poslední parametr funkce. Třetí součástí tripletu je zpráva, tedy akce, jež má být s datovou strukturou v předané kategorii provedena, např. úprava konfigurace DS (obvykle pomocí MSG_SET) a její čtení (MSG_GET).

Rozhraní TWAIN specifikuje sedm komunikačních stavů. Každý stav je vymezen specifickou třídou operací, a tedy omezuje dostupné triplety, přičemž není-li uvedeno jinak, jeden z nich umožňuje přechod do následujícího stavu a jeden návrat do stavu předchozího. Tyto stavy včetně přechodových mechanismů jsou ukázány na obrázku 5. V úvodním stav 1 není navázáno spojení mezi aplikací a jinou komponentou rozhraní TWAIN. Pomocí operací závislých na použitém operačním systému je načten DSM. Načtením DSM se dostáváme do stavu 2. Zde je provedeno zpřístupnění funkce DSM_Entry(), taktéž závislé na platformě, a pomocí tripletu DG_CONTROL/DAT_PARENT/MSG_OPENDS je DSM otevřen. Ve stavu 3 je sezení TWAIN ustaveno, a je tedy možné vybrat DS (v režii aplikace či pomocí dialogu v DSM) a otevřít jej pomocí DG_CONTROL/DAT_IDENTITY/MSG_OPENDS. Stav 4 je určen k volbě nastavení otevřeného datového zdroje, který je následně povolen kombinací parametrů DG_CONTROL/DAT_USERINTERFACE/MSG_ENABLEDS. Je-li to aplikací umožněno, ve stavu 5 zobrazí DS své specifické GUI, v opačném případě závisí na implementaci aplikace, jakým způsobem a zda vůbec zprostředkuje ovládání sezení. V obou případech DS indikuje připravenost k přenosu zasláním zprávy MSG_XFERREADY aplikaci, čímž je spojení posunuto do stavu 6. Aplikace nyní získává informace o obrázku připraveném k přenosu a kombinací DG_IMAGE/DAT_IMAGE???XFER/MSG_GET (podoba DAT_IMAGE???XFER záleží na zvolené metodě přenosu) zahájí jeho přenos ve stavu 7. Forma přenosu závisí na nastavené přenosové metodě. Ukončení přenosu dat a návrat do stavu 6 je dosažen tripletem DG_CONTROL/DAT_PENDINGXFERS/MSG_ENDXFER. Nyní je možné opakovat stejný postup pro další dostupné obrázky nebo se vrátit do stavu 5 pomocí DG_CONTROL/DAT_PENDINGXFERS/MSG_RESET, přičemž návrat je automatický, pokud již žádné další obrázky nejsou dostupné. Zakázání DS je provedeno, při zobrazení GUI povoleného DS jen po přijetí zprávy MSG_CLOSEDSREQ, užitím DG_CONTROL/DAT_USERINTERFACE/MSG_DISABLEDS. Postupný návrat ze stavu 4 do stavů 3 a 2 umožňují: DG_CONTROL/DAT_IDENTITY/MSG_CLOSEDS a DG_CONTROL/DAT_PARENT/MSG_CLOSEDSM. Úplné ukončení spojení (stav 1) je dosaženo uvolněním DSM, které stejně jako načtení závisí na použitém operačním systému.

Obrázek 5: Stavy TWAIN (podle [1])

Čtení a změna nastavení parametrů přenosu a samotného DS jsou prováděny pomocí DG_CONTROL/DAT_CAPABILITY/MSG_??? (např. MSG_GET, MSG_SET). Standard TWAIN definuje velké množství těchto voleb s pouhými několika povinnými.

Pro přenos dat jsou specifikovány tři metody, přičemž první dvě musí podporovat každý DS: nativní (native), do paměti (buffered memory) a do souboru (disk file). Zde se budeme zabývat jen povinnými metodami. Nativní metoda (výchozí zvolená) je z pohledu aplikace nejjednodušší, ve stavu 6 je zaslán požadavek DG_IMAGE/DAT_IMAGENATIVEXFER/MSG_GET a přenesen kompletní obrázek ve formátu závislém na platformě (BMP na Windows). Z pohledu DS se však jedná o komplexní operaci, neboť musí alokovat velké množství paměti pro celý obrázek a provést případné konverze dat do výsledného formátu. Na druhou stravu však nehrozí špatná interpretace obrazových dat na straně aplikace, protože DS zná všechny formáty, které jím spravované zařízení produkuje. Přenos do paměti všechny tyto starosti přenáší na aplikaci, jež ale má lepší informace o konečném využití obrázku, a může tak přenos optimalizovat. Ve stavu 4 je parametr ICAP_XFERMECH pomocí zprávy MSG_SET (viz čtení a změnu nastavení výše) nastaven na TWSX_MEMORY k indikaci právě metody přenosu do paměti. Dále je nutné v jednom ze stavů 4 až 6 získat podporované velikosti cílové sekce paměti zavoláním DG_CONTROL/DAT_SETUPMEMXFER/MSG_GET. Triplet DG_IMAGE/DAT_IMAGEMEMXFER/MSG_GET zahájí ve stavu 6 přenos, stejný triplet je používán i v 7. stavu, dokud není indikován konec přenosu. Tato metoda přenáší obraz buďto po celých řádcích (stripe), nebo v obdélníkových výřezech (tile). Uspořádání obrazových bodů, jejich barva, případná komprese a ostatní vlastnosti obrázku je nutné zjistit před jeho sestavením pomocí parametrů DS, přenosová metoda do paměti je nedefinuje.

3.3 Windows Image Acquisition

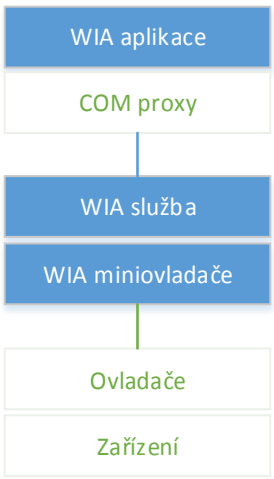
Technologie Windows Image Acquisition (dále jen WIA; [2] [17]) je aplikační rozhraní pro přístup aplikací ke skenerům, digitálním kamerám a fotoaparátům. Jedná se o proprietární technologii společnosti Microsoft přítomnou pouze v jejích operačních systémech Windows. WIA staví na modelu COM [18] (rozhraní pro vytváření binárních znovupoužitelných komponent umožňující meziprocesovou komunikaci). V současné době existují dvě verze rozhraní, 1 a 2. Verze 1 je spojena s operačními systémy Windows XP a staršími, Vista a novější obsahují verzi 2. Práce s oběma variantami je prakticky stejná, liší se jen názvy některých struktur (na konec názvu přidáno „2“) a způsob přenosu dat. Druhá verze navíc přináší více možností nastavení. Komponenty starší verze mohou být užity bez problémů pod verzí novou díky zpětné kompatibilitě.

Podobně jako v případě TWAIN sestává WIA ze tří hlavních komponent: aplikace, služby a ovladače zařízení (viz obrázek 6). Služba spravuje veškeré nainstalované ovladače zařízení, které zpřístupňuje aplikacím v systému COM komponentou správce zařízení WIA („WIA device manager“), což je struktura IWiaDevMgr nebo IWiaDevMgr2 podle zvolené verze rozhraní. V aplikaci, která je konzumentem dat, vytvořená instance správce umožňuje získat seznam dostupných zařízení a číst jejich

vlastnosti, především identifikátor, název a typ. Získání a vybrání zařízení lze provést plně v režii aplikace zavoláním metody `EnumDeviceInfo()` a poté `CreateDevice()`, která již vytvoří instanci objektu `IWiaItem` (`IWiaItem2`) vybraného zařízení. Druhou možností je využití metody `SelectDeviceDlg()`, která zobrazí dialog pro výběr skeneru (kamery, fotoaparátu, …). Objekt takto vytvořený zprostředkuje další komunikaci mezi aplikací a zařízením bez přítomnosti správce. Jde o kořenový uzel hierarchického stromu položek, které reprezentují jednotlivé složky zařízení (např. plocha skeneru, podavač). Složky zařízení podobně jako zařízení samotné poskytují po přetypování (užitím metody `QueryInterface()`) na typ `IWiaPropertyStorage` vlastnosti – nastavení formátu výsledného obrázku, jeho rozlišení, způsob přenosu apod.

Přenos dat počíná vytvořením objekt typu `IWiaDataCallback`, jež poskytuje cílový objekt pro data (typ `IStream`) a řídí samotný přenos. Nyní je vybrán zdroj dat z hierarchického stromu (`IWiaItem`/`IWiaItem2`). Následný postup se liší dle použité verze. V první verzi rozhraní je zdroj přetypován na `IWiaDataTransfer` a přenos zahájen zavoláním jeho metody `idtGetData()`. Druhá verze operuje s typem `IWiaTransfer` a metodou `Download()` (nahrávání dat do zařízení metodou `Upload()` není v kontextu skenerů nutno rozvíjet). V obou případech je metodě předán objekt typu `IWiaDataCallback` k řízení přenosu.

Ovladač zařízení je v případě WIA označen jako „miniovladač“ („minidriver“). Jde o ovladač v uživatelském prostoru, který poskytuje most mezi rozhraním WIA a plnými ovladači zařízení, které jsou zavedeny v režimu jádra a přímo ovládají připojený hardware. Ovladač zařízení WIA je komponentou COM, která poskytuje rozhraní `IWiaMiniDrv` a `IStiUSD`. První verze rozhraní umožňuje vytvoření tzv. „mikroovladače“ („microdriver“) – zjednodušeného ovladače plochého skeneru. V rozhraní WIA je k tomuto účelu přítomen obecný miniovladač plochých skenerů, který množství komunikace rozhraní řeší ve vlastní režii, a umožňuje tak zjednodušenou architekturu mikroovladače. Tento typ implementuje pouze tři funkce: `MicroEntry()` (zpracovává příkazy z miniovladače), `Scan()` (čte data ze zařízení a předává je miniovladači) a `SetPixelWindow()` (nastavuje skenovací oblast).



Obrázek 6: Zjednodušená architektura WIA (podle [2])

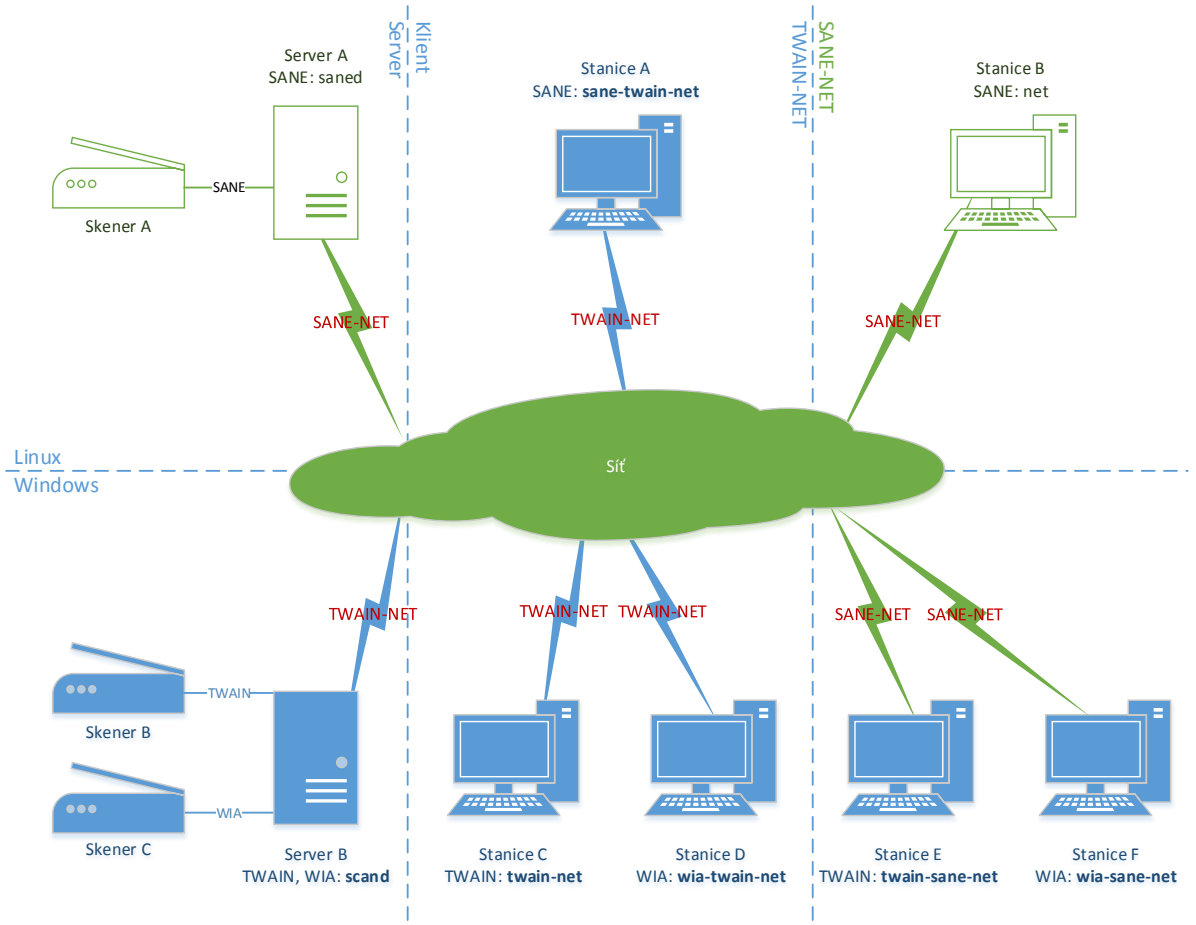
4 Návrh

V kapitole 3 byla představena tři různá aplikační rozhraní pro operační systémy Windows (TWAIN, WIA) a Linux (SANE). Nyní bude proveden návrh systému pro sdílení skenerů, které jsou k těmto rozhraním připojeny, po počítačové síti typu TCP/IP.

4.1 Architektura systému

Začněme nejprve linuxovými operačními systémy. Tato platforma využívá pro skenování služeb rozhraní SANE. Toto rozhraní je plně síťově transparentní a definuje síťový protokol. Standardní distribuce také obsahuje frontend `saned` a backend `net`, které komunikaci po síti zajišťují. Z toho důvodu není vhodné na linuxových platformách implementovat k zpřístupnění skenerů další službu, ale pouze využít službu stávající. Z hlediska klienta, resp. backendu, se také nic v případě komunikace SANE↔SANE nemění. Pro zajištění spojení se světem operačních systémů Windows však bude nutné vyvinout nový backend `sane-twain-net`, který zajistí překlad rozhraní z jiného systému do SANE.

Operační systém Windows pro připojení skenerů poskytuje rozhraní TWAIN a WIA, které jsou velmi podobné z pohledu dostupných možností nastavení skenerů, avšak žádný síťový protokol nespecifikují, popř. nejsou ani plně síťově transparentní. Pro zpřístupnění skenerů tedy bude nutné navrhnout novou službu `scand` a síťový protokol. Z důvodu zjednodušení architektury, velké podobnosti obou rozhraní TWAIN a WIA a autorovy lepší obeznámenosti s prvně zmíněným rozhraním, bude navržen síťový protokol pouze pro rozhraní TWAIN, označme jej `TWAIN-NET`. Veškerá síťová komunikace se skenery připojenými k rozhraní WIA tak bude překládána do rozhraní TWAIN a poté přenesena. Nevýhodou tohoto rozhodnutí je vyšší režie, a tedy i vyšší nároky na službu `scand`. Pro aplikace rozhraní TWAIN budou vytvořeny dva nové DS `twain-net`, který umožní síťovou komunikaci protokolem `TWAIN-NET`, a `twain-sane-net` pro přístup ke skenerům pomocí protokolu SANE, dojde v něm tedy k překladu TWAIN↔SANE. Obdobná situace nastává u aplikací WIA, kde pro komunikaci se skenery připojenými v OS Windows bude využit nový miniovladač `wia-twain-net`, který zajistí překlad WIA↔TWAIN a využití protokolu `TWAIN-NET`. Spojení s linuxovými skenery pak pomocí protokolu SANE zajistí `wia-sane-net` (WIA↔SANE). Ilustraci navržené architektury lze nalézt na obrázku 7.



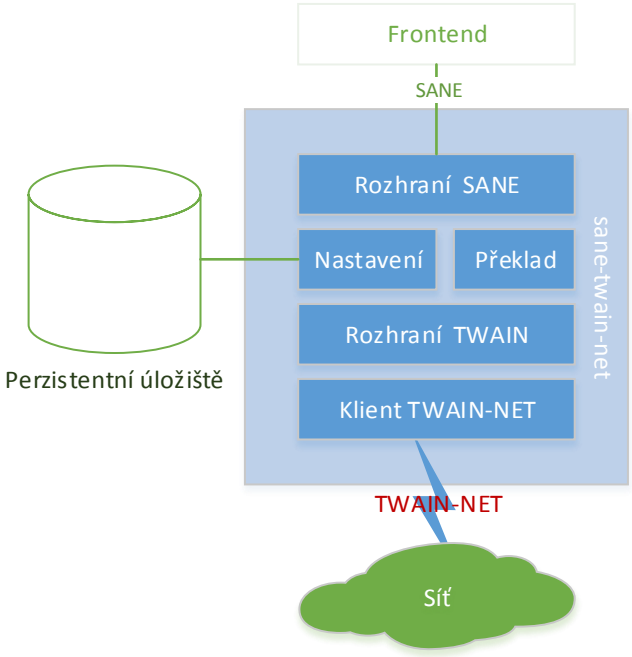
Obrázek 7: Architektura systému

4.2 Linux

Pro linuxové systémy je nutno navrhnout pouze jediný backend rozhraní SANE, v obrázku 7 označený jako `sane-twain-net`, který umožní spojení se skenery připojenými v operačních systémech Windows. Na obrázku 8 je ukázáno schéma tohoto backendu. Jedná se v podstatě o překladač funkčních volání SANE↔TWAIN a klienta protokolu `TWAIN-NET`. Nastavení spočívá pouze v zadání seznamu adres serverů a případně uživatelského jména a hesla, podobně jako je tomu u backendu `net`.

4.3 Windows

V případě systémů společnosti Microsoft je nutno pokrýt všechny možnosti vlastním řešením. Jedná se o dvě rozdílná rozhraní TWAIN a WIA, pro něž je nezbytné vytvořit server a klienty. Server využívá pouze protokolu `TWAIN-NET`, klienti musí podporovat protokoly dva: `TWAI-NET` a SANE.



Obrázek 8: Schéma backendu SANE

4.3.1 Klient TWAIN

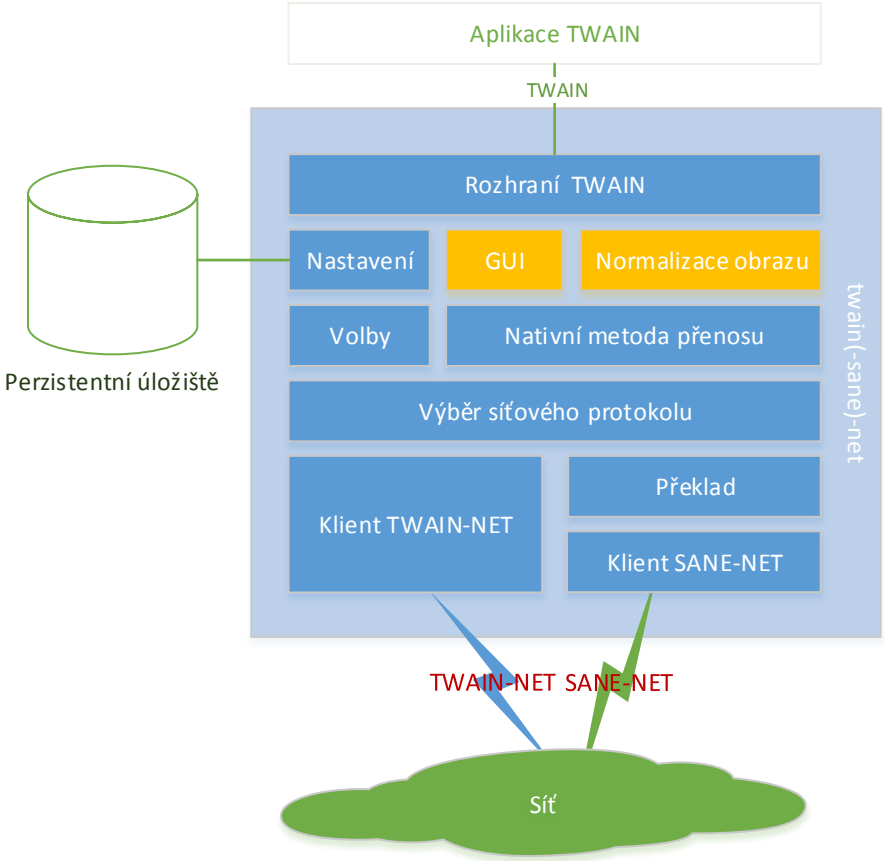
Obrázek 7 ukazuje, že je třeba vytvořit dva DS pro komunikaci dvěma rozdílnými síťovými protokoly. Avšak vzhledem k faktu, že drtivá většina funkčnosti je stejná a jen síťové protokoly se liší, lze vytvořit jediný DS, který zastane obě funkce a pouze vybere protokol podle typu zpřístupněného vzdáleného skeneru. V případě SANE proběhne nejprve překlad TWAIN↔SANE.

V architektuře TWAIN představuje DS právě jedno zařízení, proto jméno každé instance DS `twain-net` musí obsahovat identifikátor, podle kterého je vybrán správný vzdálený skener. Tedy v nastavení pod tímto označením je přítomna adresa serveru, uživatelské údaje (jméno, heslo), typ protokolu (SANE, `TWAIN-NET`), lokální název DS a identifikátor vzdáleného skeneru.

Součástí každého DS je GUI, které aplikace může zobrazit. Od tohoto rozhraní je očekávána těsná spolupráce a znalost skeneru, jež ovládá. Tento projekt si však klade za cíl vytvořit generický DS s GUI, které je naprosto nezávislé na skeneru, čímž jsou jeho možnosti značně omezeny. Pro zmírnění dopadů bude GUI přesunuto do nezávislé knihovny a umožněno tak vytvořit a použít pro speciální typy skenerů rozhraní více specializované. V nastavení pak přibude volba GUI. Pro potřeby zobrazení náhledu skenovaného obrazu bude připojená knihovna grafického rozhraní průběžně informována o přijatých obrazových datech z nativní metody.

Z důvodu vysokých paměťových a výpočetních nároků na straně serveru nebude nativní metoda přenosu obrazu protokolem `TWAIN-NET` podporována. Tato povinná přenosová metoda tedy musí být implementována přímo v klientském DS překladem zbývající paměťové metody. Vzhledem k velkému počtu možných formátů dat paměťové metody, bude v DS přítomen normalizátor, který data z metody převede na jednotný formát nativní metody. Normalizátor implementovaný v této práci bude pracovat

pouze s několika nejčastějšími formáty, a tedy podobně jako v případě GUI, bude i tato komponenta umístěna v oddělené knihovně pro možnosti podpory dalších typů obrazových dat. Implementací nativní přenosové metody v DS také dochází k povinnosti zpracovávat volbu přenosové metody lokálně v tomto DS. Obrázek 9 ilustruje navrženou architekturu zdroje.

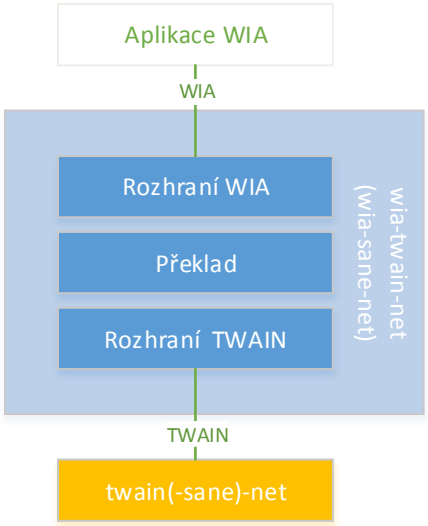


Obrázek 9: Schéma zdroje TWAIN

4.3.2 Klient WIA

Jako v případě DS pro rozhraní TWAIN, i miniovladač rozhraní WIA je třeba realizovat pro dva zmíněné síťové protokoly. Vzhledem k faktu, že síťová rozhraní jsou řešena již v DS `twain-net`, lze miniovladač navrhnout jako pouhý překladač WIA↔TWAIN. Tím mírně vzrostou nároky na výpočetní výkon, odměnou však bude nepoměrně menší komplexnost miniovladače, a tedy i menší možnost výskytu chyb, což je u klienta, kde není předpokládán nepřetržitý běh stejné instance programu, preferovanější možnost. Navíc přidání nového komunikačního protokolu či opravy stávajících, budou řešeny pouze na jediném místě – v DS.

Na obrázku 10 je zobrazeno navržené schéma miniovladače. Přeložené příkazy aplikace jsou předávány dále odpovídajícímu DS. Miniovladač tedy v tomto směru vystupuje jako aplikace TWAIN. Identifikátor DS je zapsán do informačního souboru před samotnou instalací ovladače, jedná se o jeho součást, proto zde není přítomna položka pro nastavení.



Obrázek 10: Schéma miniovladače WIA

4.3.3 Server

Byly již popsány všechny varianty klientů a navrhnuta jejich řešení, nyní je čas prozkoumat možnosti serveru. Na OS Linux je použita distribuce SANE se službou `saned`, v případě Windows bude implementován server vlastní.

Server, na obrázku 7 pojmenován `scand`, je aplikace běžící na pozadí, nejlépe jako služba systému Windows. Program čeká na příchozí síťová spojení či příkazy od již připojených klientů, žádnou vlastní (nevyžádanou) aktivitu nevyvíjí.

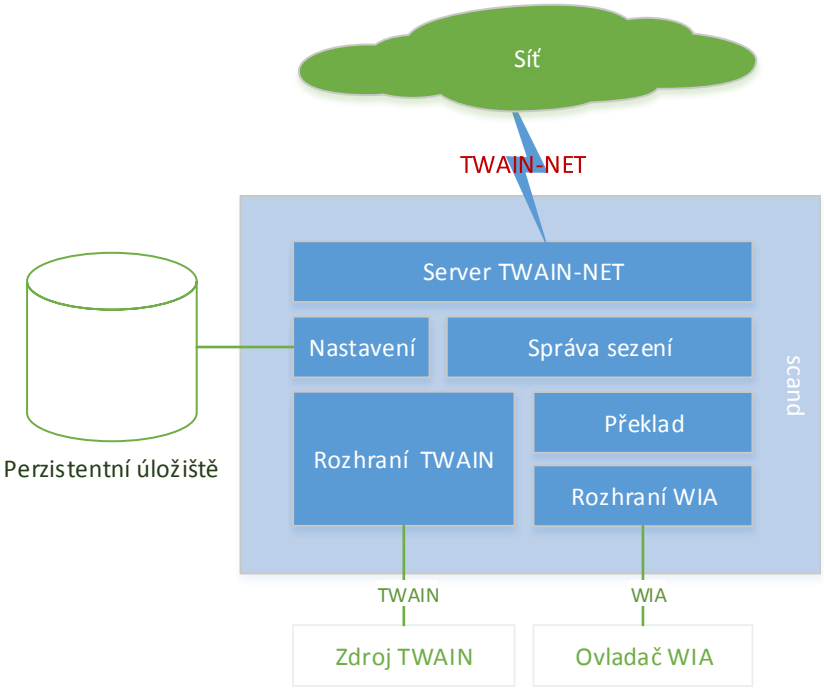
Nastavení sestává z čísla portu protokolu TCP, jež je použito k čekání na spojení, seznamu uživatelů (jména, hesla, možnost anonymního přístupu) a seznamu sdílených skenerů (typ TWAIN/WIA, identifikátor, název, výrobce, maximální počet spojení). Identifikátor skeneru sestává v případě TWAIN z kombinace názvu a výrobce, u rozhraní WIA jde o číslo přiřazené systémem instanci miniovladače.

Modul správy sezení uchovává stav protokolu jednotlivých spojení (připojeno, autentizováno/autorizováno, otevřeno spojení se skenerem) a kontroluje omezení maximálního počtu spojení ke každému skeneru dle jeho preferencí.

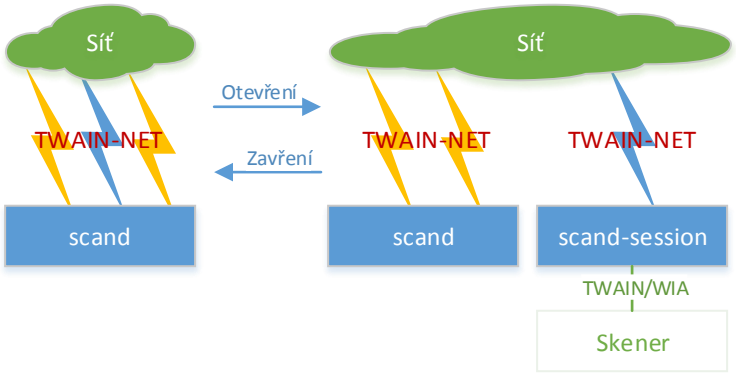
Podle typu skeneru je vybráno rozhraní TWAIN nebo WIA a může probíhat komunikace mezi vzdálenou aplikací a tímto skenerem. U skenerů připojeným k rozhraní WIA dochází k překladu WIA↔TWAIN. Schéma komponent serveru je ukázáno na obrázku 11.

Podobně jako v [19] je i zde po otevření skeneru vytvořen nový proces (namísto pouhého vlákna), který zajišťuje samotnou komunikaci se zařízením. Jedná se o opatření zajišťující stabilitu serveru, miniovladač či DS skeneru nemusí správně fungovat mimo hlavní vlákno aplikace a navíc může obsahovat chyby. Vytvořením nového procesu je tak ohroženo pouze toto konkrétní spojení a nehrozí pád celého serveru. Tento mechanismus je ukázán na obrázku 12, kde po otevření skeneru jedním ze tří komunikačních kanálů dojde k vytvoření nového procesu, v němž probíhá komunikace s vybraným

skenerem, a po zavření zařízení je proces ukončen. Každé spojení může mít v jednu chvíli otevřen nejvýše jeden skener.



Obrázek 11: Schéma serveru TWAIN/WIA



Obrázek 12: Otevření a zavření skeneru

4.4 Síťový protokol

V [19] byl specifikován aplikační protokol, který je možné použít i v této práci. Avšak zbytečná složitost a nevhodnost pro asynchronní zpracování, jež byla způsobena absencí položky s délkou požadavku a následně i odpovědi, jej pro další používání diskvalifikují. Bude tedy navržen nový, jednodušší aplikační síťový protokol.

Protokol z [19] vyžadoval transportní protokol TCP, zde postačí libovolný spolehlivý protokol, který zachovává pořadí doručených zpráv. Pro jednodušší implementaci asynchronního zpracování

každá zpráva, není-li řečeno jinak, začíná položkou s délkou zbývající části. Čísla jsou reprezentována polem bajtů v pořadí od nejvýznamnějšího bajtu po nejméně významný. Textové řetězce jsou přenášeny jako řetězce ordinálních hodnot znaků (čísel) bez ukončovacího nulového bajtu.

Po navázání spojení na přenosové vrstvě zašle klient serveru úvodní zprávu s identifikací protokolu a jeho nejvyšší podporovanou verzí. Server odpoví stavovým číslem (v pořádku, chyba, potřeba autentizace) a vybranou verzí protokolu, obvykle jde o nejnižší verzi z nejvyšších možných verzí podporovaných klientem i serverem. Tento dokument specifikuje verzi nula.

Vyžaduje-li server autentizaci, klient ji provede zasláním uživatelského jména, hesla zašifrovaného jednosměrnou šifrou a kryptografické soli užité při šifrování hesla. Uživatelské jméno i kryptografická sůl jsou předcházeny svojí délkou. Šifra hesla má délku vždy stejnou, je užita šifra SHA-256 z [20].

Po úspěšné autentizaci či ihned, pokud server autentizaci nevyžaduje, je klient připojen i na aplikační vrstvě a smí zasílat serveru dotazy. Tyto sestávají ze dvou typů: zjištění dostupných DS a provádění příkazů TWAIN (od otevření DS výše a zpět po jeho uzavření; příkazy nižších stavů jsou plně v režii serveru). Dotaz nulové délky je možné použít pro udržování spojení, server jej kromě restartu odpovídajících časovačů ignoruje.

V následujících podkapitolách jsou přesně definovány jednotlivé požadavky klienta a odpovědi serveru. Zprávy jsou reprezentovány tabulkami, kde sloupec odpovídá zasílané položce, první řádek určuje význam položky a druhý řádek počet bajtů. Je-li v prvním řádku položka uvedena v jednoduchých uvozovkách, jedná se přímo o přenášený znak. Také číslo v prvním řádku vyjadřuje přímo přenášenou hodnotu. Text v hranatých závorkách značí odkaz na hodnotu uvedené položky. Pole je uvedeno textem ve složených závorkách v prvním řádku a počtem položek ve druhém řádku. Následuje prázdný sloupec, po kterém je uvedena šablona položek v poli. Za poslední položkou šablony je přítomen taktéž prázdný sloupec. Delší zprávy jsou uvedeny v několika dvouřádkových tabulkách bezprostředně pod sebou.

4.4.1 Obálka zpráv

Veškeré zprávy jsou přenášeny v této obálce:

DÉLKA ZPRÁVY	ZPRÁVA
4	[DÉLKA ZPRÁVY]

- DÉLKA ZPRÁVY – počet bajtů zprávy ZPRÁVA.
- ZPRÁVA – zpráva, formát různých typů zprávy definován dále.

4.4.2 Stavové kódy

Protokol definuje tyto stavové kódy (položky STAV):

Hodnota	Význam
0	Bezchybné provedení. Jde-li o ustavovací zprávu, není požadována autentizace.
1	Chyba, je vyžadována autentizace. V případě ustavovací zprávy není považováno za chybu. Při autentizaci indikuje špatné přihlašovací údaje.
253	Nebylo provedeno ustavení spojení dle 4.4.3.
254	Chybný formát zprávy.
255	Obecná chyba.

Stavové kódy zde neuvedené by se v komunikaci neměly vyskytnout. Reakce na neznámý stav je stejná jako při obecné chybě. Chybové kódy indikují nedefinování veškerých po nich následujících polí (tj. nemusí být ani poslány).

4.4.3 Ustavení spojení

Požadavek

0	‚t‘	‚w‘	VERZE	ZEMĚ	JAZYK	MAJ. VERZE	MIN. VERZE	SKUPINY
1	1	1	1	2	2	2	2	4

- VERZE označuje verzi protokolu klienta.
- ZEMĚ a JAZYK jsou kódy země původu a jazyka aplikace TWAIN na straně klienta.
- MAJ. VERZE a MIN. VERZE reprezentují v řadě majoritní a minoritní verzi rozhraní TWAIN připojené aplikace.
- SKUPINY je číslo identifikující podporované skupiny dat aplikace (logické „nebo“ z DG_* a popř. DF_APP2).

Odpověď

STAV	VERZE
1	1

- VERZE označuje verzi protokolu vybranou serverem s ohledem na verzi klienta.

4.4.4 Autentizace

Požadavek

1	BAJTŮ UŽIV. JMÉNA	UŽIVATELSKÉ JMÉNO	BAJTŮ KRYPT. SOLI	KRYPT. SŮL	OTISK HESLA
1	4	[BAJTŮ UŽIV. JMÉNA]	4	[BAJTŮ KRYPT. SOLI]	32

- KRYPT. SŮL – kryptografická sůl použitá při vytvoření otisku hesla; doporučená délka v rozmezí 10 až 32 bajtů.
- OTISK HESLA – otisk uživatelského hesla získaný jako otisk SHA-256 řetězce vzniklého spojením hesla a kryptografické soli v tomto pořadí.

Odpověď

STAV
1

4.4.5 Dostupné zdroje

Požadavek

2
1

Odpověď

STAV	POČET ZDROJŮ	{ZDROJE}		IDENTIFIKÁTOR	BAJTŮ NÁZVU
1	4	[POČET ZDROJŮ]		16	4

NÁZEV	BAJTŮ VÝROBCE	VÝROBCE	ARCHITEKTURA	
[BAJTŮ NÁZVU]	4	[BAJTŮ VÝROBCE]	1	

- IDENTIFIKÁTOR – unikátní identifikátor zdroje.
- NÁZEV – název modelu připojeného skeneru či název DS.
- VÝROBCE – výrobce skeneru či jeho DS, kombinace NÁZEV a VÝROBCE musí být unikátní pouze v rámci jednoho serveru.
- ARCHITEKTURA – architektura, resp. počet bitů architektury, zdroje (32 nebo 64).

4.4.6 Příkazy TWAIN

Příkazy TWAIN sdílejí stejný formát hlavičky pro požadavek a odpověď. Není-li uvedeno jinak, argumenty jsou přenášeny jako posloupnost svých položek (prohledávání do hloubky) v požadavku i v odpovědi. Výjimky jsou uvedeny dále. Protokol nespecifikuje mechanismy pro oznámení připravenosti serveru na přenos klientovi. Přesun ze stavu 5 do 6 je implicitně předpokládán po obdržení nechybové odpovědi při požadavku na povolení vzdáleného datového zdroje. Server tento přesun musí zajistit ve vlastní režii; dojde-li k chybě, musí zůstat ve stavu 4.

Přenášené typy argumentu

- `DAT_CAPABILITY`,
- `DAT_EXTIMAGEINFO`,
- `DAT_IDENTITY`,
- `DAT_IMAGEINFO`,
- `DAT_IMAGELAYOUT`,
- `DAT_IMAGEMEMXFER`,
- `DAT_PALETTE8`,
- `DAT_PENDINGXFERS`,
- `DAT_SETUPMEMXFER`,
- `DAT_STATUSUTF8`,
- `DAT_USERINTERFACE`,
- `DAT_XFERGROUP`.

Hlavička požadavku

255	DG	DAT	MSG	OBSAHUJE DATA	DATA
1	4	2	2	1	

- DG – kategorie operace (`DG_*`).
- DAT – typ argumentu (`DAT_*`).
- MSG – akce k provedení (`MSG_*`).
- OBSAHUJE DATA – příznak, zda jsou přítomna data argumentu.
- DATA – data argumentu, formát a délka závisí na typu DAT tohoto argumentu.

Hlavička odpovědi

STAV	VÝSLEDEK	STATUS	DATA STATUSU	OBSAHUJE DATA	DATA
1	2	2	2	1	

- VÝSLEDEK – výsledek požadované operace (`TWRC_*`).
- STATUS – upřesnění výsledku operace (`TWCC_*`).
- DATA STATUSU – doprovodná data upřesnění výsledku operace (obvykle vynulováno).
- OBSAHUJE DATA – příznak, zda byla data argumentu změněna a jsou přítomna.
- DATA – data argumentu, formát a délka závisí na typu argumentu.

DAT_CAPABILITY/TW_CAPABILITY

.Cap	.ConType	OBSAHUJE DATA	.hContainer
2	2	1	

- OBSAHUJE DATA – zda jsou přenášena data kontejneru `.hContainer`; nejsou-li, `.hContainer` má hodnotu nulového ukazatele.
- `.hContainer` – data kontejneru, jedna ze struktur `TW_ARRAY`, `TW_ENUMERATION`, `TW_ONEVALUE`, `TW_RANGE`. Položky proměnného typu v těchto strukturách jsou přenášeny jako nastavený typ, ne jako typ, který je uvedený v definici struktury.
- Žádná volba („capability“) nepoužívá typ TWTY_HANDLE, klient proto tento typ vyhodnocuje jako neznámý. Zavržené („deprecated“) typy `TWTY_STR1024` a `TWTY_UNI512` také nejsou podporovány.

Požadavek DAT_EXTIMAGEINFO/TW_EXTIMAGEINFO

.NumInfos	{TW_INFO}		.Info.InfoID	
4	[.NumInfos]		2	

Odpověď DAT_EXTIMAGEINFO/TW_EXTIMAGEINFO

.NumInfos	{TW_INFO}		.Info.InfoID	.Info.ItemType	.Info.NumItems
4	[.NumInfos]		2	2	2

.Info.ReturnCode	OBSAHUJE DATA	{DATA}		.Info.Item		
2	1	[.Info.NumItems]		[.Info.ItemType]		

- `.NumInfos` v požadavku a odpovědi musí obsahovat stejnou hodnotu.
- OBSAHUJE DATA – zda je přenášeno pole {DATA}; není-li, `.Info.Item` má hodnotu nula.
- `.Info.Item` – pole jednotlivých hodnot v `TW_INFO` tak, jak jej definuje standard TWAIN v [1]. Význam a způsob přenosu jedné hodnoty záleží na `.Info.ItemType`.
- Nemá-li server možnost zjistit velikost dat položky, nastaví `.Info.NumItems` a OBSAHUJE DATA na 0 a `.Info.ReturnCode` na `TWRC_INFONOTSUPPORTED`. Toto nastává v případech, kdy `.Info.ItemType` obsahuje neznámý typ či u `TWTY_HANDLE` nelze (nebo server tak není uzpůsoben) zjistit velikost z ostatních položek.
- `TWTY_HANDLE` se známou velikostí je přenášeno jako řetězec se 4bajtovým prefixem udávajícím délku dat v bajtech.

Požadavek DAT_IDENTITY

IDENTIFIKÁTOR
16

- Data musí být přítomna jen při `MSG_OPENDS`.
- IDENTIFIKÁTOR – identifikátor požadovaného zdroje získaný při dotazu na dostupné zdroje.

Odpověď `DAT_IDENTITY`

Žádná data TWAIN nejsou přenášena.

`DAT_IMAGEINFO/TW_IMAGEINFO`

Standardní přenos. Výjimkou je pole `BitsPerSample`, jehož počet přenesených položek odpovídá hodnotě `SamplesPerPixel`. Tedy není vždy přeneseno všech osm prvků pole, obvykle pouze tři (RGB).

Požadavek `DAT_IMAGEMEMXFER/TW_IMAGEMEMXFER`

.Memory.Length
4

- `.Memory.Length` – velikost vyhrazené paměti pro příjem obrazových dat v bajtech.

Odpověď `DAT_IMAGEMEMXFER/TW_IMAGEMEMXFER`

.Compression	.BytesPerRow	.Columns	.Rows	.XOffset	.YOffset
2	4	4	4	4	4

.BytesWritten	.Memory.TheMem
4	[.BytesWritten]

- `.Compression` – typ komprese dat (`TWCP_*`).
- `.BytesPerRow` – počet bajtů na řádek včetně případné vycpávky („`padding`“).
- `.Columns` – počet přenesených obrazových sloupců.
- `.Rows` – počet přenesených obrazových řádků.
- `.XOffset` – hodnota levého okraje přenesených obrazových dat.
- `.YOffset` – hodnota horního okraje přenesených obrazových dat.
- `.BytesWritten` – velikost přenesených obrazových dat v bajtech.
- `.Memory.TheMem` – obrazová data.

`DAT_PALETTE8/TW_PALETTE8`

.NumColors	.PaletteType	{TW_ELEMENT8}		.Colors. Channel1	.Colors. Channel2	.Colors. Channel3	
2	2	[.NumColors]		1	1	1	

- `.Compression` – typ komprese dat (`TWCP_*`).
- `.BytesPerRow` – počet bajtů na řádek včetně případné vycpávky („`padding`“).

Požadavek DAT_STATUSUTF8/TW_STATUSUTF8

.Status.ConditionCode	.Status.Data
2	2

Odpověď DAT_STATUSUTF8/TW_STATUSUTF8

.Status.ConditionCode	.Status.Data	.Size	OBSAHUJE DATA	.UTF8string
2	2	4	1	[.Size] - 1

- OBSAHUJE DATA – zda je přenášeno pole `.UTF8string`; není-li, `.UTF8string` má hodnotu nulového ukazatele.

DAT_USERINTERFACE

Nejsou přenášena žádná data.

Řetězce (TW_STR*)

DÉLKA	DATA
4	[DÉLKA]

- DÉLKA – počet bajtů řetězce bez ukončovacího nulového bajtu.
- DATA – data řetězce bez ukončovacího nulového bajtu.

5 Implementace

V předchozí kapitole 4 byla navržena celková architektura systému pro sdílení skenerů po síti. Byly popsány jednotlivé části na operačních systémech Windows a Linux a specifikován komunikační síťový protokol `TWAIN-NET`. Účelem této kapitoly je implementace navrženého systému v programovacím jazyce C++ za využití knihoven Boost a frameworku Qt. Zmíněný framework bude využit především pro vytvoření grafického uživatelského rozhraní. Možnosti návrhových byly konzultovány s [21]. Cesty k souborům uvedené v následujících kapitolách jsou relativními cestami ke kořenové složce projektu.

5.1 Podpůrné knihovny

Jazyk C++ ve spojení s knihovnou Boost poskytuje množství tříd, struktur, funkcí aj. pro řešení rozmanitých problémů při vývoji softwaru. Jde však především o problémy obecného charakteru, přičemž řešení konkrétnějších záležitostí je ponecháno na uživateli. V této kapitole budou popsány právě takové případy a jejich implementační řešení pro potřeby této práce.

5.1.1 SHA-256

Framework Qt poskytuje implementaci algoritmu SHA-256 z [20], avšak její využití je podmíněno použitím dynamické knihovny o velikosti několika jednotek MB v závislosti na architektuře, překladači a dalších nastaveních. Vzhledem k použití v síťovém protokolu, a tedy i na serveru, kde není vhodné zvyšovat bezdůvodně paměťovou náročnost, a také k relativní jednoduchosti, bylo rozhodnuto vytvořit implementaci vlastní.

Soubor `sha256/sha256.hpp` obsahuje naprogramovaný algoritmus. Zapsání samotné funkce SHA-256 odpovídá algoritmům popsaným v [20]. Hlavní veřejnou funkcí je `Hash::sha256()`, která vrací výsledné bajty hašovací funkce. Pro kompatibilitu s PHP[4] a textový výstup je poskytnuta funkce `Hash::sha256Hex()`, která surový výstup převádí na textový řetězec, v němž každé dva znaky odpovídají hodnotě náležitého bajtu v šestnáctkové soustavě, např. „ff“ odpovídá bajtu s hodnotou 255.

5.1.2 Dynamické knihovny a sdílené objekty

Manažer datových zdrojů TWAIN je umístěn v dynamické knihovně, backendy SANE ve sdílených objektech (mohou být i ve statických). Práce s oběma typy je pro naše účely totožná, liší se však

[4] PHP: Hypertext Preprocessor, funkce hash s algoritmem sha256 a vypnutým surovým („raw“) výstupem, více na http://php.net/manual/en/function.hash.php.

podpůrné funkce. I zde pouze Qt poskytuje vlastní implementaci, kterou zavrhujeme ze stejných důvodů jako v kapitole 5.1.1.

Implementace je umístěna v souboru `shared-library/sharedlibrary.hpp`. Operační systémy Windows používají dynamické knihovny a funkce `LoadLibrary()`, `FreeLibrary()` a `GetProcAddress()` pro otevření a zavření knihovny a získání adresy pojmenovaného symbolu v otevřené knihovně (např. funkce nebo data). V případě Linuxu a sdílených objektů jde o funkce `dlopen()`, `dlclose()` a `dlsym()`. Třída `SharedLibrary` tyto rozdíly skrývá a poskytuje jednotné programovací rozhraní.

Při vytvoření instance je knihovna (objekt) otevřen, při destrukci instance dochází automaticky k jejímu uzavření. Nemůže dojít k situaci, kdy uživatel zapomene zdroj uzavřít. Díky tzv. move sémantice v C++11 (a emulaci pomocí `Boost.Move`) lze starost o otevřený zdroj přesunout do jiné instance třídy. Programovací vzor *Resource Acquisition Is Initialization* [5] (zkráceně RAII) je použit i u dalších tříd, které pracují se zdroji vyžadujícími uvolnění, a nebude již u nich dále popisován.

5.1.3 Procesy

Spouštění procesů a práce s nimi je další oblastí závislou na operačním systému. Rozdíly mezi oběma podporovanými systémy jsou větší než v případě sdílených objektů a dynamických knihoven, budou proto popsány hlouběji.

Souborem určeným k inkluzi je `process/process.hpp`, který definuje rozhraní – funkci `Process::start()`. Tato funkce je přetížena tak, aby umožňovala spuštění nezávislého procesu („detached“) a procesu, při jehož ukončení je provedena požadovaná akce.

Windows poskytuje k vytváření procesů funkci `CreateProcess`. Vytvoření závislého či nezávislého procesu je indikováno pouze změnou parametru. Závislý proces lze pak sledovat (resp. asynchronně počkat na jeho ukončení) pomocí knihovny Boost.Asio, přesněji instance třídy `boost::asio::windows::object_handle`. Funkce `GetExitCodeProcess()` slouží k získání návratového kódu procesu. Zdrojový soubor: `process/process_win32.hpp`.

Linux (unixové systémy) používají k vytvoření nového procesu funkci `fork()`. Nový proces je téměř identickou kopií původního a oba pokračují v provádění stejného kódu. Hlavní rozdíl spočívá v návratové hodnotě `fork()`, která je nulová pro dětský proces a nenulová pro rodičovský. Pro spuštění jiného programu je nutné zavolat funkci z rodiny `exec()` (např. `execve()`). Vzhledem k tomu, že rozhraní `Process::start()` očekává příkaz v jediném řetězci, je použita funkce execl(), která spouští systémový shell (`/bin/sh`) a v parametru mu předává požadovaný příkaz.

Fakt, že rodičovský proces musí počkat na dětský pomocí `wait()`, `waitpid()` nebo podobných funkcí, jinak dětský proces nikdy plně neskončí, a také že ukončení rodiče znamená ukončení potomka, mírně komplikuje vytvoření nezávislého procesu. Namísto přímého vytvoření požadovaného procesu je vytvořen další proces jako mezistupeň. Tento mezistupeň rychle zapne zadaný proces a okamžitě

skončí (rodičovský proces na něj čeká). Tím dojde ke změně rodiče požadovaného procesu na `init` či jiný systémový proces, který vždy běží a automaticky čeká na všechny své dětské procesy. Původní rodičovský proces může pokračovat bez dalšího čekání.

Manipulace závislého procesu je podobná jako u Windows, namísto handle je čekáno pomocí `Boost.Asio` na signál `SIGCHLD`, který je zaslán při zastavení potomka. Pomocí `waitpid()` je přečten stav potomka, čímž je plně ukončen. Implementace pro Linux a unixové systémy je přítomna v souboru `process/process_unix.hpp`.

5.1.4 Jednoduchá šifra

Služba serveru i klient TWAIN ukládají svá nastavení do perzistentního úložiště, které může být, v závislosti na nastavené politice, přečteno uživatelem. Pro ochranu uložených hesel vytvoříme jednoduchou šifrovací knihovnu. Cílem není zabránit získání hesla odhodlaným pokročilým uživatelem, postačí ochrana před náhodným přístupem.

Pro naše potřeby postačí použít jako šifru logickou operaci exkluzivní nebo (XOR) v šifrovacím módu CBC[5] popsaném v [22]. Jako klíč je zvolena náhodná posloupnost bajtů a uložena do zdrojového souboru, tedy jde o konstantu. Před šifrováním je vygenerován náhodný inicializační vektor o velikosti jednoho bloku, který je umístěn na začátek výstupu. Vstup je zarovnán podle PKCS #7 [23]. Samotná šifra je provedena dle následující rovnice:

$$C_i = P_i \oplus K \oplus C_{i-1} \tag{1}$$

Symbol C_i označuje současný výstupní zašifrovaný blok, P_i současný vstupní blok, K je klíč a C_{i-1} předchozím výstupním blokem. Pro $i = 0$ je C_{i-1} inicializačním vektorem.

Rozhraní této jednoduché šifry je zveřejněno v souboru `simple-crypt/simplecrypt.hpp`. Soubor `simple-crypt/simplecrypt.cpp` pak obsahuje vlastní implementaci.

5.1.5 Base64

Do textového souboru nebo proudu nelze přímo zapisovat binární data (např. výstup obecné šifry), je nutné je převést. Pro tuto operaci lze využít funkci `Hash::sha256Hex()` z kapitoly 5.1.1, avšak její konverzní poměr je 2, tedy každý bajt na vstupu je zakódován jako dva bajty na výstupu. U potenciálně větších dat je takový poměr nevhodný. Kódování Base64 popsané v [24] s poměrem $\frac{4}{3}$ vyhovuje. Bylo rozhodnuto využít existující implementaci z [25], která byla upravena do čistě hlavičkové podoby („header-only implementation“). Upravený zdrojový kód knihovny se nachází v jediném souboru `_ext/base64/base64.hpp`, který je určený k inkluzi.

[5] Cipher Block Chaining

5.1.6 Užitečné funkce a třídy

Následující sekce popisují funkce a třídy, které vzhledem ke své rozmanitosti a jednoduchosti nejsou přímou součástí žádné knihovny, ale jsou použity napříč celým projektem. Nadpis sekce převedený na malá písmena s koncovkou `.hpp` uvádí implementační soubor v adresáři `utils`.

ArrayFlat

Standard C++11 přichází s podporou konstantních výrazů (klíčové slovo `constexpr`) a uživatelsky definovaných literálů. Díky tomu lze zapsat pole konstantní velikosti, které je vytvořeno přímo v době překladu. Struktura `Utils::ArrData` umožňuje za využití další novinky v C++11 – šablon s proměnným počtem argumentů – takové pole vytvořit i z položek, jejichž počet je menší než velikost tohoto pole. Přebytečné položky jsou inicializovány na výchozí hodnotu.

Pomocí struktury `Utils::IndexList` převzaté z [26] je získán seznam indexů vstupního pole. Rekurzivní struktura `Utils::ArrFlatImpl` převezme ze vstupního pole poslední položku a zároveň dědí sama sebe s polem o tuto položku menším. Při přístupu pomocí indexu pak vrací svoji položku, pokud index odpovídá velikosti vstupního pole zadaného v průběhu vytvoření o jedna menším, jinak předá řízení zděděné struktuře. Není-li index nalezen, je vrácena výchozí hodnota typu pole. Struktura `Utils::ArrDataImpl` tímto nakonec naplní výstupní pole konečné velikosti všemi položkami vstupního pole a přebytečné položky nastaví na výchozí hodnotu. Je-li vstupní pole větší než výstupní, překladač generuje chybovou hlášku. Pro překladače, které nepodporují C++11 a novější, je poskytnuta implementace, která vytváří výstupní pole za běhu pomocí kopie.

Díky tomuto lze definovat uživatelské literály, které mají předem danou velikost, avšak mohou být v době překladu vytvořeny z jiných literálů různých velikostí. Konkrétní využití bude představeno v dalších kapitolách.

EnumClass

Klíčové slovo `enum` poskytuje možnost vytvořit datový typ výčet, jehož možné hodnoty jsou omezeny na seznam pojmenovaných konstant, které jsou umístěny do jmenného prostoru, kde byl výčet definován. C++11 zavádí tzv. třídní výčet („enum class“), díky kterému jsou položky definovány ve jmenném prostoru výčtu, nikoliv mimo něj. Dále je možné specifikovat datový typ (a tím velikost konstant), nad kterým je výčet vytvořen.

Pomocí maker `UTILS_ENUM_CLASS_BEGIN()` a `UTILS_ENUM_CLASS_END()` je možné ve starších specifikacích C++ emulovat funkcionalitu třídního výčtu. `UTILS_ENUM_CLASS_TYPE_BEGIN()` a `UTILS_ENUM_CLASS_TYPE_END()` umožňují navíc uvést implementační datový typ výčtu. Nepatrným omezením při použití takovýchto výčtových typů je nutnost použít metodu `value()` v případech, kde je vyžadován číselný typ (např. konstrukt `switch`). V parametru šablony, kde je očekávána konstanta, se namísto zadaného názvu typu použije vnitřní výčtový typ `Value`.

IntIO

Zde jsou implementovány struktury pro zápis/čtení libovolných číselných datových typů (o velikostech 1, 2 a 4 bajty) do/z vyrovnávací paměti. Při zápisu jsou čísla převedena tak, aby nejvýznamnější bajt byl první a nejméně významný poslední. Čtení provádí inverzní převod. Pro překladače s podporou šablon s proměnným počtem parametrů jsou poskytnuty struktury, které umožňují číst a zapisovat libovolný počet čísel v jednom funkčním volání.

NoAllocator

Standardní kontejnery při změně velikosti inicializují nové položky na výchozí hodnotu, odebírané položky volají svůj destruktor. Obě tyto operace jsou zbytečné, pokud je kontejner použit jako vyrovnávací paměť nad vestavěným číselným datovým typem. Pomocí změny alokátoru („allocator“, přímý český překlad „přidělovač“ není použitelný) na zde implementovaný `Utils::NoAllocator` lze těmto operacím zabránit. Jde o standardní alokátor, jehož metoda `destroy()` a metoda `construct()` bez dodatečných parametrů jsou prázdné.

Nullptr

Pouhá definice klíčového slova `nullptr` z C++11 pro starší překladače jako makra s hodnotou nula.

PackSize

Struktura `Utils::PackSize` umožňuje pomocí rekurze zjistit celkovou velikost libovolného počtu parametrů šablony. Hlavní využití je u šablon s proměnným počtem parametrů, které potřebují získat jejich celkovou velikost. Jde o šablonu, výpočet je proveden v době překladu.

ReaderBase

Zde zapsaná třída `Utils::ReaderBase` poskytuje metody ke čtení čísel a určeného počtu bajtů ze síťové vyrovnávací paměti. Před každým čtením je pomocí virtuální metody `checkSize()` zkontrolováno, zda lze požadovaná data přečíst, v záporném případě je generována výjimka. Data je možné přečíst, pokud jejich velikost nepřesahuje velikost vyrovnávací paměti zmenšenou o celkovou velikost již přečtených dat. Tímto je zamezeno chybě přetečení paměti. Třída je určena jako základní rodičovská třída pro implementace nad konkrétním protokolem.

StdoutSuppress

Aplikace, jejíž standardní výstup musí odpovídat určitému formátu, nutně musí dočasně zakázat výstup, pokud spouští nový proces nebo volá funkce, nad nimiž nemá kontrolu.

Pomocí funkce `fopen()` je otevřen virtuální soubor NUL (Windows) nebo `/dev/null` (Linux), který veškerý vstup ignoruje. Funkce `dup()` provede vytvoření záložní kopie deskriptoru standardního výstupu a `dup2()` do původního deskriptoru nastaví v úvodu otevřený soubor (pomocí `fileno()` je získán jeho deskriptor). Nyní jsou veškeré pokusy o zápis do standardního výstupu ignorovány.

Opětovného povolení je dosaženo dalším voláním funkce `dup2()`, která však nyní jako standardní výstup nastaví záložní kopii vytvořenou funkcí `dup()`. Na Windows jsou použity funkce se znakem podtržítko („_“) jako předponou.

Pro jednodušší použití je tato funkcionalita zapouzdřena do třídy `Utils::StdoutSuppress`. Při vytvoření instance je ve výchozím nastavení standardní výstup zakázán, destrukce znamená opětovné povolení. Jsou také poskytnuty metody `stop()` a `start()`, které umožňují explicitní nastavení výstupu.

StrictAlias

Programovací jazyk C++ umožňuje pracovat s pamětí určitého datového typu jako s typem jiným. Pokud je takto pracováno s nekompatibilními typy, překladač obvykle zobrazí upozornění. Je-li zapnuta odpovídající optimalizace, může docházet i k chybnému zpracování programu.

Úkolem funkce `Utils::alias_cast` je tyto optimalizace a upozornění v místech použití zakázat. Je využito faktu, že jakýkoliv ukazatel lze implicitně přetypovat na ukazatel typu `void`, který je pro účely přetypování kompatibilní s libovolným jiným typem ukazatele. Funkce je určena jako náhrada za standardní operátor `reinterpret_cast` nad ukazateli nekompatibilních typů.

SyncClientBase

Společné operace síťových klientů zahrnují otevření spojení a jeho uzavření, čtení a zápis dat. Tyto operace mohou trvat dlouhou dobu, proto je záhodno nastavit maximální dobu čekání. Třída `Utils::SyncClientBase` tyto operace poskytuje nad knihovnou `Boost.Asio`. Blokující operace jsou simulovány pomocí asynchronních, neboť `Boost.Asio` nepodporuje přímo nastavení čekací doby, která je tak simulována časovačem. Samotné čtení a zápis do vyrovnávací paměti je ponecháno na uživateli, jsou však poskytnuty metody `transferExactly()` a `flush()`, které v prvním případě tuto paměť naplní definovaným počtem bajtů zaslaných druhou stranou spojení[6] a ve druhém ji druhé straně zašlou. V obou případech je použit časovač omezující dobu nečinnosti.

TcpServer

Knihovna `Boost.Asio` také obsahuje funkcionalitu potřebnou k naslouchání na příchozí síťová spojení. Ve třídě `Utils::TcpServer` je naslouchání na příchozí spojení protokolu TCP nad IPv4 i IPv6 spojeno do jednotného rozhraní. Naslouchání na zadaném portu je provedeno asynchronně. Podtřídy již pouze implementují virtuální metodu `newConnection()`, jíž je předán objekt popisující nové spojení.

ThrowDestructor

Generování výjimek v destruktoru je potenciálně nebezpečná operace. Byla-li již výjimka vygenerována v jiné části kódu a destruktor při uvolnění zásobníku vygeneruje další, program skončí s chybou. Není možné, aby v jednom čase existovaly dvě výjimky. Z tohoto důvodu standard C++11

[6] Přesněji je vytvořen objekt, který takové naplnění za využití funkce `boost::asio::async_read()` umožní.

označuje destruktory klíčovým slovem `noexcept`, které indikuje, že výjimka z této funkce není nikdy vyhozena. Dojde-li však k takové situaci, že by jinak výjimka vyhozena byla, program končí s chybou. Pro povolení vyhazování výjimek z destruktoru je nutné jej označit kombinací `noexcept(false)`. Makro `THROWING_DESTRUCTOR` umožňuje označení destruktoru v C++11 i zpětnou kompatibilitu se staršími verzemi standardu. Navíc je jednodušší takto označené destruktory v kódu najít, protože `noexcept(false)` se může vyskytovat i u jiných funkcí a metod.

TransferSize

Funkce `boost::asio::async_read()` a `boost::asio::async_write()` očekávají objekt určující konec čtení/zápisu. Ve jmenném prostoru `boost::asio` jsou definovány tři funkce, které takové objekty vytvářejí: `transfer_all`, `transfer_at_least` a `transfer_exactly`. První indikuje čtení či zápis veškerých dostupných dat, druhá minimálních počtu bajtů a třetí přesného počtu. Tyto objekty však nevyhovují v případech, kdy je využit časovač operace. Čtení nebo zápis velkého počtu dat může trvat déle než časovač dovoluje, i když jsou data přenášena. Třídy `Utils::TransferAtLeast` a `Utils::TransferExaclty` imitují chování druhého a třetího případu, dodatečně ale při každé úspěšné operaci volají metodu `resetTimeout()` rodičovského klienta. Tím je umožněno provádět čtení a zápis velkých objemů dat a operace, které do určeného času data nepřenesou, budou stále přerušeny.

WoW64Check

Programy vytvořené pro 32bitovou architekturu mohou být spuštěny i na 64bitové verzi systému Windows. Pro určité operace je výhodné či přímo nutné, aby si architektury programu a systému odpovídaly. Díky funkci `isWoW64()`, jejíž kód byl převzat z [27], je rozlišení 32bitového a 64bitového systému možné. 64bitová aplikace je vždy spuštěna na 64bitovém systému.

WriterBase

Třída `Utils::WriterBase` je obdobou `Utils::ReaderBase`. Jak již název napovídá, tato varianta slouží k zápisu dat do vyrovnávací paměti. Číselné hodnoty a pole bajtů jsou zapisovány na konec paměti, přičemž tato je vždy rozšířena tak, aby mohla pojmout právě zapisovaná data. Při destrukci instance je zavolán funkční objekt (obvykle zprostředkuje posílání dat po síti), který byl předán při vytvoření. Toto je jediný destruktor v celém projektu, který smí vyhazovat výjimky.

5.2 Framework SANE++

Distribuce SANE obsahuje hlavičkový soubor `sane.h` (lze nalézt v adresáři `_ext/sane`), který definuje datové typy, struktury a konstanty a deklaruje funkce uvedené v manuálu [3]. Hlavičkový soubor je určen pro jazyk C, konstanty jsou zapsány jako makra, taktéž převodní funkce. Datové typy `SANE_Word`, `SANE_Fixed`, `SANE_BOOL` a `SANE_INT` lze bez jakékoliv konverze použít jako stejný typ, i když jejich sémantika je odlišná, např. `SANE_Fixed` je 32bitové číslo, jehož prvních šestnáct bitů je použito jako celá část čísla s pevnou desetinnou čárkou a druhých šestnáct bitů značí desetinnou část (tedy sémantická hodnota tohoto dílu odpovídá zapsané dělené číslem 65536). Stejná sémantická hodnota tedy pokaždé odpovídá jiné sekvenci bitů, avšak překladač nevygeneruje žádné upozornění. Struktury `SANE_Device` a `SANE_Option_Descriptor` obsahují položky s typem ukazatele, čímž je umožněno do nich odkazy na dynamicky alokovanou paměť, jež musí být po použití uvolněna. Uvolnění takové paměti je přenecháno uživateli, a tedy může dojít k opomenutí. Konečně z důvodu použití jazyka C jsou hodnoty výčtových typů `enum` umístěny v globálním jmenném prostoru.

Bylo rozhodnuto vytvořit jednoduchý binárně kompatibilní framework SANE++, který tyto nedostatky odstraní nebo aspoň minimalizuje a také ulehčí vytvoření frontendu a backendu. Framework byl umístěn do adresáře `sanepp`, veškeré jeho součásti jsou obsaženy ve jmenném prostoru `Sanepp`.

5.2.1 Datové typy

Vstupní soubor sanepp.hpp zahrnuje („include“) celý framework a definuje typy `Byte`, `Word`, `Int` a `Handle` (stejné jako v oficiálním souboru, avšak s explicitním uvedením velikosti základového datového typu u číselných typů). Typ `Bool` byl implementován jako třída nad typem Word s implicitními operátory konverze z/do vestavěného typu `bool`, je tak zaručena typová bezpečnost a zároveň binární kompatibilita s původním typem `SANE_Bool`. Řetězcové typy nejsou definovány, namísto toho je použit standardní typ `char`.

Číslo s pevnou desetinnou čárkou je nyní reprezentováno třídou `Fixed`. Pro zjednodušení práce byly vytvořeny implicitní konverzní operátory z/do vestavěného typu float a aritmetické a logické operátory sčítání, odčítání, násobení, dělení a porovnání (rovno, nerovno, menší, menší nebo rovno, větší, větší nebo rovno).

Pro funkce pracující s verzí byla vytvořena třída `Version`, která zajišťuje čitelnější rozhraní pro získání majoritního, minoritního a dodatečného („build“) čísla verze. Makra `SANE_VERSION_*` určená pro práci s číselným typem již nejsou potřeba.

Výčtové typy `SANE_Action`, `SANE_Constraint_Type`, `SANE_Frame`, `SANE_Value_Type` a `SANE_Unit` byly převedeny na třídní výčtové typy `Action`, `Constraint`, `Frame`, `Type` a `Unit` emulované pomocí maker EnumClass. Typ `Status` (`SANE_Status`) byl převeden pomocí stejné nebo podobné techniky, avšak bez využití preprocesorových maker z důvodu přidání speciální funkcionality. Obsahuje navíc operátor konverze na typ `bool`, který vrací pravdivou hodnotu, právě když hodnota

instance odpovídá položce `Status::Good` (`SANE_STATUS_GOOD`), která jediná označuje bezchybný stav. Z příznakových maker `SANE_CAP_*` a `SANE_INFO_*` byly vytvořeny třídy `Capability` a `Info`, jež zlepšují typovou bezpečnost a přímo poskytují metody k zadání příznaků a jejich zjištění.

Struktura `SANE_Parameters`, nyní `Parameters`, byla pouze převedena na nové rozhraní a namísto přímého přístupu k prvkům byly poskytnuty metody.
Z důvodu rozlišení základového typu typů `Int` a `Fixed` musí být struktura `SANE_Range` přepsána zvlášť pro každý typ. Šablony jazyka C++ ale umožňují vytvořit jedinou implementaci Detail::Range, která je následně použita jako základový typ (s patřičně nastaveným vnitřním typem) pro typy `IntRange` a `FixedRange`.

Poslední dvě struktury, `SANE_Device` a `SANE_Option_Descriptor`, reprezentují případy, kdy obsažené ukazatele mohou ukazovat na data v téměř libovolné části paměti. Je ponecháno na posouzení uživatele, zda paměť je po použití nutno uvolnit či nikoliv. Třída `Device` taktéž své ukazatele nevlastní, pouze jsou namísto přímého přístupu k prvkům použity metody. Pro účel uvolnění paměti po použití byla vytvořena třída `DeviceOwner`. Pomocí šablony je uživatelem při vytvoření nadefinováno, které ukazatele budou po použití automaticky uvolněny, a ukazatelé předáni. Velikost `DeviceOwner` a rozložení položek odpovídá `Device`, tedy je možné ji použít v případech, kdy je očekávána instance třídy `Device` (je poskytnut operátor implicitní konverze).

Případ tříd `Option` a `OptionOwner` je na první pohled stejný. Použití unie (klíčové slovo `union`) nad třemi rozdílnými ukazateli věc komplikuje. Řešením je definice `Option` podobně jako v případě `Device`, avšak `OptionOwner` namísto zachování podobného rozvržení nyní přímo obsahuje objekt `Option` a pro jeho každý ukazatel je vytvořena položka indikující, zda je vlastněn či nikoliv. Díky operátoru implicitní konverze lze stále instance této třídy použít na místě `Option`.

5.2.2 Frontend

Aplikace SANE, koncový frontend, se připojuje k backendu `dll`, který spravuje ostatní backendy. Ke zjednodušení této činnosti je v souboru `dll.hpp` přítomna třída `Dll`. Zapouzdřuje otevření `dll` pomocí `SharedLibrary` z kapitoly 5.1.2 a zpřístupňuje funkce SANE, které nepracují s otevřeným zařízením, ve formě metod objektu. O práci se zařízeními se stará třída Session, díky které nemusí uživatel při každém volání funkce specifikovat handle otevřeného zařízení.

5.2.3 Backend

Vytvoření backendu s sebou nese několik úskalí, která nejsou z manuálu [3] zřejmá. Po kontrole zdrojového kódu oficiálního backendu `dll` je zjištěno, že nejsou hledány funkce SANE s prefixem „`sane_`“, ale „`sane_<název_objektu>_`“, kde „`<název_objektu>`“ odpovídá názvu souboru (mimo případné předpony „lib“ a přípony) sdíleného objektu backendu. Každý backend musí také obsahovat

funkci pro konverzi stavu na řetězec, obvykle se jedná o konstanty, které se za běhu nemění. V neposlední řadě je zde nutnost spravovat jednotlivá sezení pro každé otevření zařízení.

Pro skrytí těchto problémů a další zjednodušení práce byla implementována třída `BackendFromThis` (resp. `BackendFromThisCustom`, která umožňuje nastavit vlastní funkční objekt pro konverzi stavu na řetězec). Jedná se o třídní šablonu určenou k dědění, při kterém uživatel pouze zadá jméno své třídy a funkční objekty pro inicializaci a uzavření backendu a pro získání seznamu připojených zařízení. Práce s handle otevřených sezení je řešena interně, uživatel pouze implementuje (čistě) virtuální metody, které operují již nad samotným zařízením. Po vytvoření třídy backendu je nezbytné použít makro `SANEPP_STATIC_DEFINES`, které definuje funkce rozhraní SANE tak, aby volaly příslušné statické metody backendu. Příklad použití lze nalézt v komentáři na začátku souboru.

5.3 Síťový protokol SANE

V kapitole 4.3.1 byl navrhnut datový zdroj TWAIN. Pro komunikaci se zařízeními připojenými pomocí SANE na linuxových operačních systémech je využit síťový protokol z [3]. Výchozí implementace v jazyce C je zatížena licencí GNU General Public License[7] verze 2 nebo vyšší, proto bude vytvořena implementace vlastní. Zdrojový kód se nachází v adresáři `sane-net/protocol` a je umístěn ve jmenném prostoru `SaneProtocol`. Kompatibilita se staršími standardy jazyka C++ není zajištěna.

Hlavním souborem je `protocol.hpp`, který definuje výčtové typy označující uspořádání bajtů při přenosu obrazových dat a identifikační čísla jednotlivých funkcí SANE pro potřeby jejich vzdáleného volání. Tento soubor zpřístupňuje veškeré další třídy.

Ke čtení odpovědí na požadavky klienta byla v souboru `reader.hpp` vytvořena třída `Detail::Reader`, která dědí `Utils::ReaderBase`. Čtení jednotlivých struktur v odpovědích je realizováno kombinací metod z bázové třídy. Obsah odkazů výstupních objektů je umístěn ve vyrovnávací paměti, pouze před návratem z metody proběhne uspořádání bajtů. Případné pole odkazů na řetězce v popisovači volby (`Option`) je umístěno v pomocném objektu. Tímto je zajištěno, že uživatel nemusí uvolňovat paměť. Nevýhodou je platnost takových hodnot pouze do dalšího přečtení odpovědi (popř. pouze do zapsání požadavku, je-li paměť sdílena) a nutnost hluboké kopie, pokud je třeba platnost delší. Síťový protokol SANE nezasílá v hlavičce celkovou velikost odpovědi, nelze tedy při prvním čtení určit velikost. Původní metoda `checkSize()` by tak při každém pokusu o čtení generovala výjimku. Jejím přepsáním („override“) a dodáním speciálního funkčního objektu, jenž zajišťuje příjem bajtů od serveru, je dosaženo chování opačné. Při každém pokusu o čtení, pro který nedostačují data, je funkční objekt zavolán a data od serveru získána. Toto však může mít za následek zvětšení oblasti vyrovnávací paměti, které může být realizováno alokací nového místa a kopií původních položek. V kombinaci s ukládáním obsahu výstupních ukazatelů jde o nebezpečnou operaci,

[7] Více na https://www.gnu.org/copyleft/gpl.html.

která může mít za následek neočekávané chování programu nebo jeho ukončení s chybou. Řešením je při čtení nastavovat hodnoty odkazů pouze na pozice položek ve vyrovnávací paměti od jejího začátku. Jakmile je čtení celé odpovědi dokončeno, a nemůže tedy nastat přesun paměti, je veškerým ukazatelům hodnota zvětšena o hodnotu ukazatele na první prvek paměti.

Zápis požadavků je umístěn do třídy `Detail::Writer` dědící třídu `Utils::WriterBase`. Metody určené k zápisu jednotlivých typů požadavků pouze zapisují položky do vyrovnávací paměti. Jejich implementaci, která je k dispozici v souboru `writer.hpp`, není potřeba zevrubně popisovat, neboť neobsahuje žádné pokročilé postupy jako třída `Detail::Reader` uvedená v předchozím odstavci.

Soubor `syncprotocol.hpp` obsahuje třídu `Sync::Client` (dědí třídu `Utils::SyncClientBase` popsanou v sekci SyncClientBase0), která již poskytuje veřejné rozhraní pro zasílání požadavků na server a čtení odpovědí. Tyto dvě operace jsou realizovány pomocí tříd uvedených v předchozích dvou odstavcích, zde jsou abstrahovány jako volání jedné metody pro každý typ požadavku. Pro potřeby zpracování zpráv okenního systému je možné zadat speciální funkci, jež bude při vykonávání blokujících operací volána několikrát za sekundu. Případné uživatelské rozhraní díky tomu zůstane responzivní i při dlouhotrvajících požadavcích.

Přenos obrazových dat probíhá v odděleném síťovém spojení abstrahovaném ve třídě `Sync::DataClient` (taktéž dědí `Utils::SyncClientBase`). I zde je možnost zadat speciální funkci pro potřeby okenního systému. Na rozdíl od dotazů SANE, každý přenos obrazových dat obsahuje hlavičku indikující jeho velikost, lze jej tedy přenést celý bez dalšího zpracovávání položek. Ke zjednodušení práce s daty byla vytvořena metoda `readLine()`. Jejím účelem je přečtení vždy kompletního obrazového řádku, jehož data mohou být rozprostřena do několika síťových přenosů. Přenos o nulové velikosti dat je validní, avšak pro praktické účely je jejich počet v řadě shora omezen na určenou konstantu (v současnosti nejvíce sto).

5.4 Framework TWAIN++

Standard TWAIN definuje množství datových typů, struktur a konstant. Hlavičkový soubor `twain.h` (adresář `_ext/twain`) tyto definice uvádí v jazyce C, konstanty jsou reprezentovány makry. Dynamická paměť je vytvářena a uvolňována speciálními funkcemi. Před přístupem je tuto paměť třeba uzamknout, po použití odemknout. Cílem nového frameworku TWAIN++, podobně jako u SANE++ z kapitoly 5.2, je lepší typová bezpečnost a zjednodušení práce, především s dynamickou pamětí. Veškeré zdrojové soubory jsou umístěny v adresáři `twainpp`, kód je zapsán ve jmenném prostoru `Twainpp`.

5.4.1 Základní datové typy a struktury

Hlavní hlavičkový soubor twainpp.hpp, který zpřístupňuje celý framework, definuje obdobu základních datových typů TWAIN: `TW_HANDLE`, `TW_UINTPTR`, `TW_UINT8`, `TW_UINT16`, `TW_UINT32`, `TW_INT8`,

TW_INT16, TW_INT32 a TW_BOOL. Typ Bool je stejně jako u SANE++ definován jako speciální třída, která obsahuje operátory implicitní konverze z a do vestavěného typu bool.

Čísla s pevnou desetinnou čárkou reprezentuje třída Fix32. Oproti původní struktuře TW_FIX32 jsou operátory konverze mezi vestavěnými typy s plovoucí desetinnou čárkou součástí třídy. Navíc jsou definovány aritmetické operátory (sčítání, odčítání, násobení a dělení) a logické operátory pro porovnání (větší, větší nebo rovno, menší, menší nebo rovno, rovno, nerovno). Implementace je obsažena v souboru fix32.hpp.

Souřadnice vybraného výřezu obrazu jsou obsaženy ve třídě Frame v souboru frame.hpp. Jedná se v zásadě o původní strukturu TW_FRAME, která namísto přímého přístupu k prvkům využívá metody.

Práce s různými datovými typy v dynamických strukturách vyžaduje držení explicitní informace o použitém typu. Výčtový typ Type poskytuje konstanty reprezentující všechny základní datové typy TWAIN++ a řetězce z následující kapitoly 5.4.2. Do souboru type.hpp jsou také zapsány pomocné algoritmy pro získání velikosti typu popsaného konstantou a získání odpovídajícího datového typu jazyka C++ z této konstanty při překladu (využití v šablonách).

5.4.2 Řetězce

Rozhraní TWAIN využívá, pokud není specifikováno jinak, pro řetězce pole s určenou velikostí 34, 66, 130 a 256 bajtů (typy TW_STR32, TW_STR64, TW_STR128 a TW_STR256), která jsou schopna obsáhnout řetězce až o velikosti jedna menší. Důvodem je přítomnost nulového bajtu, který označuje konec. Systém Mac OS namísto nulového bajtu na konci používá na začátku řetězce bajt, jehož hodnota představuje počet znaků.

Implementační třída Detail::Str (str.hpp) umožňuje získat délku a data řetězce nezávisle na platformě (na Mac OS jsou data stále bez nulového ukončovacího bajtu). Zadání dat za běhu je řešeno tak, aby nemohlo nikdy dojít k zápisu mino vyhrazenou paměť a nulový bajt (bajt s délkou) byl vždy přítomen. V C++11 za využití třídy Utils::ArrData uvedené v sekci ArrayFlat je třída Detail::Str literálním typem. Instance lze vytvořit již v době překladu z řetězcových literálů, jejichž velikost (včetně ukončovacího bajtu), nepřesahuje velikost výsledného objektu Detail::Str, v opačném případě je zobrazena chyba překladu. Ve veřejné části frameworku jsou pak definovány typy Str32, Str64, Str128 a Str255, které využívají Detail::Str s poli o odpovídající velikosti. Taktéž jsou přítomny operátory pro porovnání dvou řetězců se stejnou velikostí pole na rovnost a nerovnost.

5.4.3 Dynamická paměť

Standard do verze 2.0 (kromě) přímo definuje použité funkce pro alokaci, dealokaci, uzamknutí a odemknutí dynamické paměti (více v [1]). V následujících verzích tyto funkce poskytuje přímo manažer datových zdrojů. V twainpp.hpp jsou v privátní sekci umístěny globální ukazatele na tyto funkce (na systémech Windows a Mac OS je také uvedena jejich výchozí implementace). Veškeré

struktury, které pracují s dynamickou pamětí, jež může být předána druhé straně spojení, používají tyto globální ukazatele na paměťové funkce.

Ke zjednodušení uzamykání a odemykání paměti slouží instance třídy `Detail::Lock` ze souboru `lock.hpp`. Při vytvoření objektu je poskytnuté paměťové handle uzamčeno, při destrukci odemčeno. Pomocí operátorů hvězdička a šipka lze přistupovat přes instanci `Detail::Lock` přímo k datům jako k ukazateli nastaveného vnitřního typu.

Třída `Detail::Data` je obdobou `Detail::Lock`, jejíž instance je však možno vytvořit z obyčejného ukazatele nebo handle. Ukazatel je pouze obsažen, handle je při vytvoření objektu uzamčeno, při destrukci odemčeno.

Ve speciálních případech může uživatelský kód potřebovat pracovat přímo s handle paměti. Třída `Unsafe::HandleOwner` zajišťuje, že volné handle bude uvolněno při destrukci instance. Je-li zavolána metoda `release()`, handle je vráceno do výlučné správy uživatele a není automaticky uvolněno v destruktoru.

Pro obecný popis bloku paměti slouží třída `Memory` (`TW_MEMORY`, `memory.hpp`). Zvláštností této třídy je možnost specifikace, která ze stran spojení (aplikace, datový zdroj, manažer datových zdrojů) zodpovídá za uvolnění paměti, a také zda je obsažen jednoduchý ukazatel nebo handle. Je-li obsažen handle a současná strana je zodpovědná za uvolnění paměti, je tato uvolněna při destrukci objektu. V případě ukazatele je dealokace paměti plně v režii uživatelského kódu na straně vlastníka. Není-li nastaveno jinak, vytvořené instance mají jako vlastníka nastavenou současnou stranu spojení. Použitím metody `free()` lze explicitně uvolnit handle a vynulovat instanci na jakékoliv straně. Toto není konečný seznam tříd pracujících s dynamickou pamětí, ostatní případy již mají specifické použití, a budou proto uvedeny v samostatných kapitolách.

5.4.4 Výčtové typy

Tabulka 1 v příloze 1 specifikuje výčtové typy, které byly vytvořeny z maker obsahujících číselné konstanty. Tyto výčty byly vytvořeny pomocí maker EnumClass. Nejprve však uvedeme speciální výčty, které byly zapsány použitím stejné metody.

Speciální hodnoty volby typu `CapType::IJpegQuality` (`ICAP_JPEGQUALITY`, `TWJQ_`) jsou zapsány v souboru `jpegquality.hpp` ve třídě `JpegQuality`. Vlastní hodnoty lze vytvořit pomocí statické metody `JpegQuality::fromInt()`.

Volání vstupní funkce TWAIN končí vrácením chybového kódu (`TWRC_*`), který popisuje míru úspěchu nebo neúspěchu operace. Tyto hodnoty jsou umístěny do třídy `ReturnCode` (alias `RC`, soubor `returncode.hpp`), která zjednodušuje detekci chyby pomocí operátoru konverze na vestavěný datový typ `bool`. Pravdivostní hodnota je vrácena, právě když chybový kód odpovídá `ReturnCode::Success`. Ostatní nechybové kódy (např. `XferDone`, `EndOfList`) je třeba detekovat pomocí explicitního porovnání hodnot.

Není-li návratový kód dostatečně popisný, lze získat dodatečné informace dotazem na status operace. Tyto hodnoty (`TWCC_*`) jsou umístěny do třídy `Status` (`status.hpp`). I zde je operátor konverze na `bool` přítomen pro usnadnění rozhodování mezi chybou a úspěšným provedením. Tento typ není čistým výčtovým typem, neboť umožňuje zadat dodatečná data závislá na datovém zdroji. Obvyklé použití však přítomnost této položky nebere v potaz.

Standard TWAIN výslovně definuje sedm stavů rozhraní (více v kapitole 3.2). Pro lepší čitelnost při detekci a sledování současného stavu jsou vytvořeny v souboru `state.hpp` výčty `DsmState` a `DsState`. První výčet je určen ke sledování stavu manažera datových zdrojů, druhý pak ke sledování samotných datových zdrojů. Výčet popisující datové typy byl představen již v kapitole 5.4.1, proto zde nebude dále popisován, odkazujeme čtenáře do uvedené kapitoly.

5.4.5 Jednoduché třídy

Množství tříd frameworku TWAIN++ bylo vytvořeno jako pouhé obalující třídy, které namísto přímého přístupu k prvkům používají k tomuto účelu metody. Další přidaná funkcionalita je zanedbatelná nebo není dostatečně zajímavá, aby zde byla zmíněna ve větší míře. Tyto třídy jsou uvedeny v příloze 1 v tabulce 2. Třídy uvedené v tabulce 3 jsou také jednoduché, navíc však ulehčují správu dynamické paměti, která je alokována při vytvoření instance a uvolněna při destrukci.

5.4.6 Tonalita obrazu

Struktury rozhraní TWAIN `TW_RGBRESPONSE` a `TW_GRAYRESPONSE` jsou definovány tak, že obsahují pole typu `TW_ELEMENT8` o právě jedné položce. Reálný počet položek P je proměnlivý a závisí na počtu bitů na obrazový bod B, kde $0 < B \leq 8$, v jiných případech nelze tyto struktury použít:

$$P = 2^B \tag{2}$$

Předem nezadaný počet prvků pole vyžaduje využití dynamické paměti. Druhé straně spojení je předáván pouze ukazatel na data, nikoliv handle, proto lze namísto operací pro práci s dynamickou pamětí TWAIN využít standardní knihovní funkce (operátory). Třída `Detail::CurveResponse` (`curveresponse.hpp`) zajišťuje správu paměti a vytvoření pole odpovídající velikosti. Veřejné třídy `RgbResponse` a `GrayResponse`, které jsou ekvivalenty k strukturám rozhraní TWAIN, tuto třídu dědí tak, aby bylo možné rozlišit operace TWAIN pouze z typu předaných dat.

5.4.7 Paměťový přenos obrazových dat

Kromě vyžadovaného typu přenosu pomocí vyrovnávací paměti bez hlaviček, podporují některé datové zdroje také přenos realizovaný stejným způsobem, avšak s přítomnými hlavičkami pro přímý zápis do souboru. Oba způsoby využívají strukturu `TW_IMAGEMEMXFER`.

Stejně jako v předchozí kapitole 5.4.6, i zde jsou pro možnost přetížení vstupní funkce definovány dvě třídy `ImageMemXfer` a `ImageMemFileXfer`. Tyto třídy sdílí společného předka

`Detail::ImageMemXferImpl`, v němž je obsažena společná funkcionalita. Implementace je umístěna v souboru `imagememxfer.hpp`. Správu dynamické paměti zajišťuje třída `Memory` (kapitola 5.4.3).

5.4.8 Dodatečné informace o obrazu

Pro získání dalších informací o přeneseném obrázku slouží struktura `TW_EXTIMAGEINFO`, která obsahuje předem nedefinovaný počet objektů `TW_INFO`. Aplikace je zodpovědná za alokaci (a dealokaci) paměti hlavní struktury, přičemž zapíše pouze identifikační čísla položek, které si přeje získat. Datový zdroj pouze naplní jednotlivé položky. Pokud obsah položky překračuje zadanou mez (velikost ukazatele), je využita dynamická paměť, za jejíž dealokaci zodpovídá druhá strana, tedy aplikace.

Výsledná třída `ExtImageInfo` (`extimageinfo.hpp`) obaluje ukazatel na data a umožňuje přístup k jednotlivým prvkům. Použití je tedy díky abstrakci odpovídá ekvivalentu `TW_EXTIMAGEINFO`, avšak reálně je obsažen pouze odkaz, za který je instance zodpovědná. Prvky jsou realizovány jako objekty třídy `Info`, které dále abstrahují přístup k obsaženým datům pomocí `Info::Items`, neboť mohou nastat tyto tři rozdílné případy:

- Celková velikost dat je menší než velikost ukazatele a typ není `Handle`. Data jsou obsažena přímo v položce `item` (`TW_INFO.Item`), jejíž sémantika je tím změněna na pole obsaženého typu a velikosti.
- Buď celková velikost dat překračuje velikost ukazatele, nebo je obsažen typ `Handle`. V tomto případě je nutné zajistit uzamykání a odemykání handle.
- Je obsaženo pole typů `Handle` o více než jednom prvku. Manuál [1] tuto možnost explicitně neuvádí, ale ani nezakazuje.

Pro abstrakci prvního a druhého případu je použita instance `Detail::Data`, která ve třetím případě obsahuje handle na pole handlů. Celkové odstínění rozdílů je dosaženo další instancí `Detail::Data`, která je již zpřístupněna v uživatelském kódu.

Data jsou zpřístupněna pod typem, který je uveden v definici rozhraní [1]. Tohoto výsledku bylo dosaženo díky specializaci šablon a uvedení datového typu pro každou položku z výčtu `InfoId`. Není-li typ definován nebo si uživatel přeje využít vlastní typy, je možné namísto třídy `Info` použít pro přístup k položkám `ExtImageInfo` třídu `InfoRaw`.

5.4.9 Volby

Volby („capability“) jsou hlavním nástrojem k nastavení parametrů spojení TWAIN a výsledného formátu obrazu a zvuku. Programová část je zajištěna strukturou `TW_CAPABILITY`, která obsahuje (v dynamické paměti) kontejnery `TW_ARRAY`, `TW_ENUMERATION`, `TW_ONEVALUE` a `TW_RANGE`. Nyní uvedeme implementaci v TWAIN++.

Základem kontejnerů je šablona třídy `Detail::ContainerBaseRaw` (`capability.hpp`), která zajišťuje definici vnitřních typů a obsahuje ukazatel na data. Tato data zde nejsou vlastněna

(alokace/dealokace), avšak probíhá jejich uzamknutí při inicializaci a odemknutí při destrukci. Třída `Detail::ContainerBase` dědí předchozí třídu, avšak namísto explicitního uvedení typů v parametrech šablony vyžaduje zadání typu volby `CapType` (resp. `CapType::Value`). Toto je umožněno přítomností šablony `Detail::Cap`, která slouží k mapování typu volby na identifikátor vnitřního typu a (vnější) datový typ samotný[8]. První třída je rodičovskou třídou pro kontejnery `ArrayRaw`, `EnumerationRaw`, `OneValueRaw` a `RangeRaw`, jejichž vnitřní typ je určen hodnotou z výčtu `Type` (resp. `Type::Value`). K prvkům je přistupováno pouze jako k základním typům, přetypování na sémantické typy (je-li třeba) musí zajistit uživatel. Implicitní sémantické typy jsou naopak zpřístupněny kontejnery `Array`, `Enumeration`, `OneValue` a `Range`, které používají druhou zmíněnou základovou třídu, a tedy nastavují vnitřní a vnější typy jen zadáním typu volby. Obě kategorie konkrétních kontejnerů definují odpovídající rozvržení odkazované paměti a také přístupové metody.

Odkázaná dat jsou obsazena v dynamické paměti spravované třídou `Capability`. Poskytnutými statickými metodami lze vytvořit instance s libovolnými typy kontejnerů, včetně případného nastavení počtu prvků (`Array`/`ArrayRaw` a `Enumeration`/`EnumerationRaw`) a obsažených hodnot (u kontejnerů s proměnným počtem prvků pomocí inicializačního seznamu v C++11). Pro přístup ke kontejnerům v instanci slouží odpovídající nestatické metody.

5.4.10 Aplikace

Aplikacím TWAIN zjednodušuje framework TWAIN++ připojení k manažeru datových zdrojů a k jeho dostupným datovým zdrojům. Aplikační část je umístěna do souboru `manager.hpp`.

Abstrakce přístupu k manažeru a jeho správa je obsažena ve třídě `Manager`. Pomocí `SharedLibrary` (5.1.2) je v metodě `load()` načtena odpovídající dynamická knihovna `twaindsm`, není-li na 32bitovém systému Windows přítomna, je provedeno načtení starší verze `twain_32`. Následně je získána adresa funkce `DSM_Entry()`. Po otevření manažera v metodě `open()` je získána struktura popisující operace nad dynamickou pamětí (pro verzi rozhraní nižší než 2, jinak jsou použity výchozí funkce). Stav manažera je sledován a při destrukci jsou provedeny operace ke korektnímu uvolnění zdrojů.

Datové zdroje jsou reprezentovány třídou `Source`. Zde jsou uvedeny metody, které zajišťují veškeré v současnosti možné operace TWAIN (kromě uživatelsky definovaných). Přetížená metoda call() slouží jako vstup optimalizovaný pro případy vyžadující stejný počet argumentů. Pro přímé použití jsou definovány funkce, jejichž název odpovídá případu užití. Tyto funkce mají pouze minimální nutný počet argumentů. Význačné funkce, které nemají přetíženou podobu funkce `call()`, jsou tyto:

[8] Vnitřní typ dle specifikace může být pouze jeden ze základních typů (5.4.1) nebo řetězec (5.4.2). Díky mapování lze v uživatelské části pracovat s typem, který odpovídá sémantice volby. Příkladem je volba `CapType::IXferMech`, jejíž typový identifikátor je `Type::UInt16`, avšak uživatelskému kódu je vrácena hodnota typu `XferMech` namísto `UInt16`.

- `open()` – otevírá datový zdroj, poté se pokusí o nastavení funkce zpětného volání pro čtení oznámení od datového zdroje (na Windows nepovinné).
- `close()` – uzavírá datový zdroj a maže registraci případné nastavené funkce zpětného volání.
- `enable()` – povoluje datový zdroj (přesun do stavu 5).
- `disable()` – zakazuje datový zdroj (přesun do stavu 4).
- `waitReady()` – abstrahuje čekání na datový zdroj po jeho povolení (stav 5). Na Windows používá zprávy okenního systému nebo zpětné volání. Ostatní platformy používají pouze zpětné volání. Tato funkce blokuje vlákno, dokud není přijata zpráva od datového zdroje.
- `processEvent()` – funkce dostupná pouze na Windows, která umožňuje zpracovat jednu zprávu okenního systému a zjistit, zda datový zdroj nezaslal zprávu. Neblokující funkce, je však nutno volat opakovaně, dokud není zpráva od datového zdroje přijata.

I třída `Source` sleduje současný stav datového zdroje a při destrukci jej korektně uvolňuje, avšak tím není uživateli zakázáno volat metody v nekorektních stavech. Rozhodnutí o špatném stavu operace je ponecháno na datovém zdroji.

5.4.11 Datový zdroj

Framework TWAIN++ ve verzi pro datový zdroj (tj. při definovaném makru `TWPP_IS_DS`) poskytuje třídu `SourceFromThis` (popř. třídu `SourceFromThisCustom`, která navíc umožňuje zadat funkční objekt, který bude zavolán při uživatelsky definované operaci), jež je třídní šablonou určenou k dědění třídami popisujícími operace datového zdroje.

Veškerá vstupní volání, která nepracují se sezením, jsou řešena interně. Jde především o operace nastavení paměťových funkcí, získání posledního stavu operace bez sezení a získání identity zdroje (uvedena uživatelem ve funkčním objektu). Otevření sezení také spadá do této kategorie, neboť při volání ještě není sezení vytvořeno, ale je vytvořeno jako jeho následek.

V případě operací nad sezením je nejprve zkontrolována validita paměťových funkcí, na Windows je případně použit zastaralý způsob získání adresy funkce `DSM_Entry()` (knihovnou `twain_32`). Řízení je předáno instanci uživatelské třídy uvedené v parametru šablony základní třídy. Jedná-li se o operaci uzavření datového zdroje a je úspěšná nebo o neúspěšnou operaci otevření, je objekt sezení smazán. Tímto je vytvořena správa sezení (otevřených datových zdrojů). Byly použity fragmenty kódu z [19].

Pro sezení je dostupné sledování (a nastavení) současného stavu datového zdroje a stavu poslední provedené operace (`Status`). Metody pro zaslání zprávy aplikaci ve stavu 5 jsou také poskytnuty. Veškeré metody implementující operace TWAIN nebo jejich rozdělení do kategorií jsou virtuální, lze je tedy jakkoliv upravovat. Výchozí implementace zajišťuje kategorizaci operací na kontrolní, obrazové a zvukové. Ty jsou dále rozděleny podle typu datového argumentu do vlastních metod. U povinných operací (a některých dalších, viz `sourcefromthis.hpp`) dodaná implementace zajišťuje korektnost

volání v současném stavu s ohledem na vyžádanou akci (`Msg`). Poté je předáno řízení uživatelské metodě. Může-li taková operace měnit stav, je při návratové hodnotě značící úspěch stav změněn způsobem odpovídajícím akci.

Minimální implementace datového zdroje musí dodat kód pro všechny čistě virtuální funkce a užít makro `TWPP_ENTRY()`, které definuje knihovní funkci `DS_Entry()`, v právě jednom zdrojovém souboru projektu.

5.5 Síťový protokol `TWAIN-NET`

Pro potřeby následujících kapitoly bude pomocí `Boost.Asio` implementována knihovna realizující síťový protokol TWAIN navrhnutý v kapitole 4.4. Byly použity stejné postupy jako u protokolu SANE z kapitoly 5.3, které již nebudou dále popisovány. Tato a veškeré následující kapitoly vyžadují podporu standardu C++11 a jejich soubory jsou uváděny relativně k podprojektu `twain-net`. Protokol je umístěn v adresáři `protocol`, jmenný prostor je nazván `TwProtocol`.

Hlavní hlavičkový soubor `protocol.hpp` definuje stavové kódy protokolu a kategorie dotazů. Užitím tohoto souboru je zpřístupněna celá knihovna. Struktura `Source` z `source.hpp` je držitelem informací o jednotlivých datových zdrojích TWAIN nebo zařízeních WIA dostupných na serveru (dále jen „zařízení“).

Třídy `Detail::Reader` (`reader.hpp`) a `Detail::Writer` (`writer.hpp`) jsou užity ke čtení/zápisu dotazů a odpovědí z/do vyrovnávací síťové paměti. Implementace proběhla stejným způsobem jako u stejně pojmenovaných tříd pro protokol SANE z kapitoly 5.3 (jmenný prostor `SaneProtocol`). Díky použití hlavičky, která uvádí celkový počet bajtů, v dotazu i odpovědi je možné celý přenos uskutečnit naráz pomocí jediné metody a čtení je prováděno nad kompletními daty. Metoda `checkSize()` je tedy použita v původní podobě. TWAIN také definuje přesný paměťový model, všechny struktury, není-li explicitně uvedeno jinak, vlastní svoji dynamickou paměť. Proto není použit postup z implementace protokolu SANE, kdy je vyrovnávací paměť přímo odkazována ve výsledných strukturách, ale je vytvořena kopie, která je těmito strukturami vlastněna.

Klientská část protokolu je zapsána v souboru `syncprotocol.hpp`, jedná se o třídu `Sync::Client`, která, jak již název napovídá, poskytuje blokující (synchronní) operace. Implementace je také obdoba třídy `SaneProtocol::Sync::Client` protokolu SANE. Jak již však bylo uvedeno, jednotlivé odpovědi jsou přenášeny v celku a pomocí jediné univerzální metody `readResponse();` toto je zajištěno hlavičkou protokolu, která uvádí velikost dotazu v bajtech.

Server sestává ze třídy `Async::Server`, která využívá `Utils::TcpServer` k asynchronnímu naslouchání na příchozí klientská spojení, a pro správu jednotlivých příchozích spojení používá `Async::ClientConnection` (viz `asyncprotocol.hpp`). Budeme se zabývat pouze druhou jmenovanou třídou. Jedná se o abstraktní základovou třídu, která abstrahuje síťovou část. Podtřídy pouze implementují operace řešící přijaté dotazy (inicializace, autentizace, seznam zařízení, příkazy

TWAIN), které jsou reprezentovány (čistě) virtuálními metodami. Kód starající se o síťové operace je obdoba klientského, avšak pořadí čtení a zápisu je opačné. Po přijetí celého dotazu je provedena jeho kategorizace a volána odpovídající obslužná metody (u dotazu TWAIN jde o dvě úrovně). Zde dochází k přečtení dotazu, je zavolána uživatelem implementovaná metoda, po jejímž skončení je vytvořena a zaslána odpověď klientovi.

Na rozdíl od klienta jsou zde přítomny dva typy časovačů. Server také používá časovač, který omezuje dobu čekání na operaci čtení/zápisu dat z/do síťového připojení, jedná se řádově desítky sekund. Druhý, podstatně delší časovač (desítky minut i více), slouží k odhalení a ukončení nečinných spojení, která jsou po této době přerušena a objekt smazán.

Operace přenosu obrazových dat v paměti je volána často. Na každý obrázek připadá podle velikosti a barevné hloubky několik desítek až stovek (tisíců) volání v rychlém sledu za sebou. Paměť je však umístěna na haldě a musí být alokována a následně uvolněna. Starost o tuto paměť je tedy předána podtřídě, která tak může lépe rozhodnout o svých paměťových a výkonových nárocích (např. provést jedinou alokaci při začátku přenosu obrazu, pro všechny přenosy použít tuto stejnou instanci a uvolnit ji až při dokončení).

Návrh služby serveru (4.3.3) hovoří o oddělených procesech pro obsloužení dotazů nad otevřeným zařízením. Je nutné zajistit explicitní podporu na straně rodičovského i dětského procesu. Je-li v rodiči volána operace otevření datového zdroje, může uživatelský kód nastavit speciální výstupní parametr `split` na logickou hodnotu pravda. Tím dojde k **nezaslání** odpovědi klientovi a je možné předat řízení dětskému procesu, kde dojde k zavolání metody `identityJoin()`, která zašle druhé straně předanou odpověď a lze pokračovat v naslouchání na požadavky. K vrácení řízení do rodičovského procesu dojde stejným způsobem, avšak role procesů jsou vyměněny. Tedy při požadavku na uzavření datového zdroje je nastaven zmíněný parametr, odpověď v dětském procesu není zaslána a je ukončen, v rodičovském procesu je zavolána metoda `identityJoin()`. Uživatelský kód zodpovídá za detekci ukončení dětského procesu i za zavolání zmíněné metody v obou případech.

5.6 Backend `sane-twain-net`

Linuxová část projektu sestává pouze z jediného backendu rozhraní SANE. Díky knihovnám popsaným v předchozích kapitolách je již pouze řešen překlad volání SANE na volání TWAIN a v opačném směru překlad odpovědí. Základem je třída `TwainNetBackend` (`sane-backend/sanebackend.hpp`), která dědí `Sanepp::BackendFromThis` (5.2.3). Síťovou komunikaci zajišťuje `TwProtocol::Sync::Client` (5.5), který operuje nad rozhraním TWAIN (resp. TWAIN++), a vyžaduje tedy překlad.

První volanou operací backendu je vždy jeho inicializace. Zde je provedeno načtení konfigurace a hlavně nastavení paměťových funkcí frameworku TWAIN++, neboť na Linuxu nejsou zadány výchozí hodnoty.

Získání dostupných zařízení vyžaduje vytvoření dočasných síťových spojení ke všem nastaveným serverům. Autentizace je řešena za pomoci uživatelského jména a hesla zapsaných v konfiguraci. V případě neúspěchu je o zadání přístupových údajů požádán uživatel způsobem uvedeným ve specifikace SANE [3]. Konverze ze struktury `TwProcotol::Source` na `Sanepp::Device` je triviální, název zařízení kromě identifikace vzdáleného datového zdroje také pro jednodušší práci obsahuje číslo odpovídajícího serveru tak, jak je uveden v konfiguraci. Řetězce jsou vytvořeny za běhu programu, je tedy využita třída `Sanepp::DeviceOwner`, pouze typ zařízení obsahuje hodnotu statickou (popisující virtuální zařízení).

Sezení je zahájeno požadavkem na otevření zařízení. Z názvu zařízení je extrahováno číslo serveru a identifikátor vzdáleného datového zdroje, pomocí kterých je navázáno spojení se serverem a datový zdroj otevřen. Dochází také k inicializaci voleb SANE++ a jejich mapování na volby TWAIN++ (popř. jinou třídu) je zaneseno v tabulce 4 v příloze 2.

Inicializace voleb sestavuje seznam instancí třídy `Option`, které tyto volby popisují pro účely frontendu. Každé instanci je automaticky přiděleno unikátní číslo a nastavena odpovídající konverzní funkce. Díky tomu je následné získání popisovače frontendem a čtení či změna parametrů volby implementováno obecnými metodami. Tyto operace probíhají ve stavu 4 vzdáleného datového zdroje.

Získání parametrů přenosu je při prvním pohledu triviální. Z definicí obou tříd `Sanepp::Parameters` a `Twainpp::ImageInfo` je zřejmé, obsah těchto tříd je sémanticky téměř totožný. Zmíněný protějšek z frameworku TWAIN++ však není možné získat ve stavech 4 a 5, pouze 6 a 7, ale parametry SANE++ mohou být vyžádány kdykoliv při otevřeném zařízení. Řešením je využití `ImageInfo` jen v korektních stavech, v ostatních jsou použity volby `CapType::IBitDepth`, `CapType::IPixelType`, `CapType::IXResolution` a `CapType::IYResolution` a třída `ImageLayout` k získání rozměrů obrazu. Data získaná druhou metodou nemusí přesně popisovat obraz, který bude následně přenášen, avšak jedná se o lepší možnost než vrácení chyby.

Přenos obrazových dat je uveden povolením vzdáleného datového zdroje, čímž díky protokolu dojde k přesunu do stavu 6 (přesun ze stavu 5 do stavu 6 je implicitní). Nyní probíhá čtení dat. Po přenesení celého obrazu se datový zdroj nastaví postupně zpět do stavu 4.

Čtení dat představuje mírnou komplikaci z důvodu různých velikostí použitých vyrovnávacích pamětí. Nastaví-li frontend výstupní paměť menší, než je velikost přeneseného obrazového bloku (který vždy musí obsáhnout celé řádky nebo sloupce a vždy aspoň jeden), musí být nepřečtený zbytek paměti uložen do dalšího čtení. Nejprve je tedy frontentu vrácen případný zbytek, až poté je přenesen nový blok obrazových dat. Výplň („padding“) není součástí výstupních dat.

Konfigurace backendu je řešena pomocí `Boost.PropertyTree`, formát souboru je zvolen co nejpodobnější konfiguračním souborům pro backendy v oficiální distribuci SANE (více v ukázkovém souboru `twan_net.conf`). Vzhledem k obvyklému umístění knihoven (dynamických objektů) a nastavení do odlišných adresářů, je konfigurace hledána postupně v těchto adresářích, přičemž je použita první nalezená:

- adresář uvedený v proměnné prostředí („environment variable“) `SANE_CONFIG_DIR`,
- současný pracovní adresář,
- `/etc/sane.d`,
- `/usr/local/etc/sane.d`.

Změna konfigurace je ponechána na uživateli, není dodán žádný nástroj. Formát souboru je dostatečně jednoduchý, aby uživatel se znalostí angličtiny a systému Linux byl schopen úprav.

5.7 Datový zdroj `twain-net`

Datový zdroj `twain-net` je klientskou částí projektu pro systém Windows. Před popsáním jeho vlastní implementace se však pozastavíme nad poznámce ke konfiguraci v kapitole 4.3.1, která uvádí, že jméno instance zdroje je klíčem pro její získání. Takový přístup vyžaduje pro každý zpřístupněný datový zdroj vytvořit kopii implementační dynamické knihovny, která může při větším počtu zabírat nezanedbatelné místo na disku. Původní projekt [19] popisuje postup, který nároky na potřebné místo podstatně zmírňuje, zde bude použit postup podobný.

Podprojekt `source-proxy` (soubor `entry.c`) reprezentuje minimální datový zdroj, který veškeré příkazy přeposílá plnému datovému zdroji. Při prvním dotazu TWAIN je načten plný datový zdroj, získáno jméno knihovny (identifikátor konfigurační položky zdroje) a předáno zdroji. Na rozdíl od [19] je použito pouze standardní rozhraní TWAIN ve formě uživatelské operace. Veškeré požadavky jsou ihned předány otevřené instanci plného datového zdroje. Implementace je zapsána v podmnožině jazyka C++ kompatibilní s jazykem C, lze vypustit běhové knihovny C++.

Implementace datového zdroje je umístěna v adresáři `source`. Zde umístěný soubor `source.cpp` obsahuje třídu `NetSource`, která dědí `SourceFromThisCustom` (kapitola 5.4.11; použita namísto `SourceFromThis` pro možnost zadat funkční objekt ke čtení nastavení před připojením aplikace). Tato třída realizuje rozhraní datového zdroje. Oproti standardnímu zdroji je nutné sledovat dva různé stavy, lokální a vzdálený. Není-li zobrazeno GUI, jsou oba stavy identické, v opačném případě dochází k jejich rozdělení. Zobrazení GUI má za následek přechod lokálního stavu ze 4 do 5, ale vzdálený stav zůstává ve 4. Modul grafického rozhraní má tedy stále k dispozici veškeré volby nastavení, neboť pracuje se vzdáleným stavem. Lze libovolně přecházet mezi stavy, nastavovat volby a přenášet data, avšak vždy musí dojít k návratu vzdáleného stavu do 4. Až v momentě, kdy uživatel vyžádá skenování obrázku, dochází k notifikaci aplikace a přesunu lokálního i vzdáleného stavu do 6 (a dále dle instrukcí aplikace). Více o stavech TWAIN v kapitole 3.2.

Volby jsou další částí, u které je třeba pozornosti. Příloha 3 v tabulce 5 uvádí volby, které nemohou být vyhodnoceny vzdáleně nebo je výhodné je upravit či vyhodnotit lokálně. Ostatní volby jsou předávány druhé straně beze změny.

Předání dotazů klientské síťové knihovně a jejich případný překlad je implementován v podtřídách abstraktní třídy `ProxyBase` (`proxybase.hpp`, `proxybase.cpp`). Třída `TwainProxy`

(twainproxy.hpp, twainproxy.cpp) předává beze změny dotazy přímo klientu TwProtocol::Sync::Client (5.5). Pouze u operací, které snižují stav TWAIN, je při chybě sítě navrácena nechybová hodnota tak, aby aplikace mohla zdroj vždy datový zdroj uzavřít.

Složitější situace nastává u třídy SaneProxy (saneproxy.hpp, saneproxy.cpp). Zde je nejprve proveden překlad operací rozhraní TWAIN++ do SANE++, následně pomocí SaneProtocol::Sync::Client (5.3) zaslán dotaz SANE vzdálenému zařízení a konečně odpověď převedena zpět do TWAIN++. Pro přenos obrazových dat je použit klient SaneProtocol::Sync::DataClient. V příloze 3 jsou v tabulce 6 uvedeny (ne)podporované operace a v tabulce 7 volby frameworku TWAIN++.

Načtení konfigurace datového zdroje zajišťuje třída Config (config.hpp, config.cpp). K průchodu stromu nastavení je užita knihovna Boost.PropertyTree. Ke zjednodušení dalšího kódu je načtení a validace knihoven nativní přenosové metody a GUI také řešeno v této třídě. Neboť operační systém Windows ve své 64bitové verzi používá 32bitové i 64bitové aplikace TWAIN, a jsou tedy přítomny obě verze rozhraní, je konfigurační soubor umístěn pouze v adresáři 32bitové části, 64bitová část také čte tento soubor. Odpadá tím nutnost spravovat dvě rozdílné konfigurace pro každý subsystém. K rozlišení architektury v cestách k souborům knihoven je využita proměnná %ARCH%, která je automaticky nahrazena počtem bitů architektury.

5.7.1 Nativní přenosová metoda a výchozí implementace

Vzhledem k faktu, že přenosový protokol TWAIN-NET nepodporuje nativní metodu přenosu a překladač z rozhraní SANE také ne, musí být tato metoda realizována na straně klienta. K implementaci je použita paměťová přenosová metoda. Kód je rozdělen do dvou částí, které nyní popíšeme.

První část je obsažena přímo ve třídě datového zdroje. Jedná se o metodu imageNativeXfer(), která zajišťuje rozhraní nativní přenosové metody. Dochází zde k inicializaci paměťové přenosové metody a samotnému přenosu dat ve smyčce. Voláním normalizační knihovny (obrázek 9, normalizace obrazu) dochází k vytvoření výstupního obrázku. Nejprve je vytvořeno prázdné plátno, ve smyčce jsou postupně předávány nové části obrázku, poté dochází k finalizaci (tyto tři části jsou implementačními detaily knihovny) a předání výsledného obrazu volajícímu. Knihovna je inicializována při povolení datového zdroje a uzavřena při jeho zakázání.

Normalizační knihovna je dynamickou knihovnou implementující rozhraní uvedené v hlavičkovém souboru source_native_image.h (C++/TWAIN++ nebo C/TWAIN). Úkolem knihovny je vytvoření bloku paměti pro výsledný obraz v nativním formátu ve vlastní režii pomocí poskytnuté vstupní funkce TWIAIN (popř. funkcí TWAIN++), a poté postupné sestavení obrazu z jednotlivých bloků paměťové přenosové metody. Knihovna tedy přímo sestavuje obraz v nativním formátu.

Výchozí implementace je umístěna v souboru `source-native/entry.cpp`. Vytvoření bloku paměti pro výsledný obraz je provázeno získáním informací o přenášeném obrazu (`ImageInfo`). Jsou podporovány pouze barevné (RGB), šedé (resp. ve stupních šedi) a černobílé obrázky bez komprese. Obrázek musí být sestaven z jediného rámce (`PlanarChunky::Chunky`). Podporovaná barevná hloubka je 24, 8 a 1 pro barevné, šedé a černobílé obrázky. Nejsou podporovány barevné palety. Výstupní obraz ve stupních šedi obsahuje barevnou paletu o 256 prvcích, černobílý obraz obsahuje prvky dva (v obou případech barvy rovnoměrně rozprostřeny mezi černou a bílou) a hodnoty v obrazovém poli jsou indexy do těchto palet. Barevné obrázky přímo obsahují barevné hodnoty, nikoliv indexy. Do výsledného bloku paměti je zapsána hlavička obsahující odpovídající informace. [28]

Při plnění obrazu je do paměti zapisováno zdola nahoru (směr dat ve formátu BMP). Výchozí implementace podporuje pouze obrazová data zapsaná po řádcích (ne po sloupcích). Černobílá a šedá data jsou zkopírována bez úprav, u barevných dat dochází ke konverzi z RGB (pořadí složek červená, zelená, modrá) na BGR (modrá, zelená, červená; používá formát BMP).

5.7.2 Grafické uživatelské rozhraní a výchozí implementace

Každý datový zdroj musí obsahovat GUI, které je na žádost aplikace zobrazeno uživateli. GUI standardního DS je vázáno na zařízení, může tedy poskytnout specifické možnosti nastavení. Výstupem tohoto projektu je však obecný DS s obecným GUI. Pro možnost dodat GUI optimalizované pro konkrétní zařízení bylo toto přesunuto do dynamické knihovny. Jeho aplikační rozhraní je zapsáno v souboru `source_user_interface.h` (jazyky C i C++).

Na straně datového zdroje dochází k inicializaci knihovny GUI při jeho povolení s vyžádaným zobrazením GUI. Naopak uzavření GUI je provedeno při zakázání zdroje. Každé zahájení přenosu i jeho ukončení jsou oznámeny, průběžně se předává procento přenesených dat. U nativní metody je při zahájení předáno handle na blok paměti s obrazovými daty a po každém přečtení bloku paměťové metody je GUI informováno také o pozici a rozměrech nové části obrazu. Uživatelské rozhraní tedy může jednoduše implementovat zobrazení průběžného náhledu nativní metody. Při použití paměťové přenosové metody lze zobrazit pouze průběh přenosu dat.

Grafické uživatelské rozhraní vytvořené v tomto projektu je umístěno v adresáři `source-gui`. Jde o obdobu implementace z [19], kód byl však téměř celý přepsán a modifikován pro účely nového datového zdroje. Vstupní funkce GUI jsou umístěny v souboru `entry.cpp`. Dochází zde k inicializaci frameworku Qt a volání manipulačních metod komponent. Hlavní okno (`dialog.hpp`, `dialog.cpp`, `dialog.ui`) umožňuje nastavení rozlišení a barvy obrazu, je zde i možnost získání náhledu a zahájení skenování (pouze ve skenovacím módu, jinak je zobrazeno tlačítko pro uložení změn). Přenosy obrazových dat ukazují stav v indikátoru průběhu, nativní přenosy také, pokud uživatel zaškrtne příslušné pole, zobrazují průběžná obrazová data. Pro zobrazení dat je užito komponent GDI+ [29].

Získání náhledu je realizováno jako nativní přenos v malém rozlišení v režii GUI. Před zahájením přenosu se nastaví malé rozlišení (přibližně 50 až 75 DPI[9] podle datového zdroje) a po něm dochází k navrácení na předchozí hodnotu. Komponenta `Preview` (`preview.hpp`, `preview.cpp`) implementuje zobrazení náhledu. Lze zde také pomocí počítačové myši vybrat oblast obrazu, která bude ve výsledku předána aplikaci. Použití je intuitivní, dvojité kliknutí na plochu náhledu zajistí vybrání celého obrazu.

5.8 Miniovladač `wia-twain-net`

Ovladač WIA byl navržen (4.3.2) jako pouhý překladač volání mezi rozhraními TWAIN a WIA. Hlavním posláním tohoto projektu je vytvoření funkčního ovladače WIA, minimální implementace postačuje, bude tedy implementován mikroovladač.

Instance mikroovladače běží v kontextu služby správce zařízení WIA, a to neustále, implementující knihovna není načtena až v momentu připojení k zařízení jako v případě TWAIN. Operace tedy musí při dokončení uzavírat své zdroje (DSM, síť), ne až při uvolnění mikroovladače, jinak hrozí znepřístupnění těchto prostředků pro ostatní aplikace. Naše implementace (`wia-microdriver/main.cpp`) s tímto počítá. Struktura `ScanStore` obsahuje veškeré potřebné nastavení datového zdroje, seznam možných rozlišení a při přenosu dat drží celý paměťový blok včetně dodatečných informací o něm (přečtená a zbývající data, jejich velikost, počet bajtů na řádek). K naplnění struktury dochází při prvním požadavku na čtení či zápis nastavení. Operace získání a změna rozlišení (obě osy) nevyžaduje, pokud je struktura inicializována, připojení k datovému zdroji, neboť obsahuje seznam možných rozlišení. Stejná situace nastává u barevného modelu obrazu. Přečtení skenovací oblasti spojení nevyžaduje, neboť se nemění, vyžadováno je až u úpravy, kdy je nejprve zdroji nastaveno rozlišení i barevný model, a poté změna provedena.

Funkce `MicroEntry()` plní úlohu hlavního vstupního bodu, který zpracovává zadané příkazy. Tabulky 8 a 9 v příloze 3 zobrazují podporované operace, jejich význam v mikroovladači a vzájemné mapování barevných modelů WIA a TWAIN++.

Druhou důležitou funkcí je `SetPixelWindow()`, která nastavuje skenovací oblast. Také zde může dojít k inicializaci instance `ScanStore`. Po otevření datového zdroje a nastavení rozlišení, barevného modelu a barevné hloubky je zadána skenovací oblast. Při úspěšné operaci (popř. nepřesnému vyhodnocení, kdy jsou přesné hodnoty získány novým voláním) je oblast zanesena do instance `ScanStore` a také předána miniovladači. Dochází i k úpravě počtu pixelů a bajtů na řádek a počtu řádků podle oblasti.

Nyní můžeme přejít ke skenovací funkci `Scan()`, která taktéž může inicializovat objekt `ScanStore`. Při prvním volání je otevřen datový zdroj, nastaveno rozlišení, barevný model a hloubka a skenovací oblast. Poté je pomocí nativní přenosové metody získán obraz ve formátu BMP bez

[9] Body na palec („dots per inch“).

souborové hlavičky, který je uložen do instance `ScanStore`. Až do ukončení probíhá přenos tohoto obrazu do vyrovnávací paměti nadřazeného miniovladače. Obraz je přenášen po řádcích od prvního logického po poslední (formát BMP umožňuje uspořádání řádků odspodu nahoru i svrchu dolů). Kvůli internímu použití čísel s plovoucí desetinnou čárkou a zaokrouhlovací chybě může dojít k situaci, kdy miniovladač očekává jiný počet řádků nebo sloupců než je reálně získáno (jednotky). Je-li očekáváno méně řádků nebo sloupců, je přenesen tento počet. Nadbytečné řádky jsou vyplněny černou barvou, body v nadbytečném sloupci barvou předchozích bodů.

Nedílnou součástí ovladače je instalační soubor INF [30] `wia-microdriver/wiascannet.inf`. Tento soubor specifikuje 32bitový i 64bitový mikroovladač. Jedná se o šablonu, která očekává dodání popisu (jména zařízení), seznamu podporovaných rozlišení a identifikace odpovídajícího datového zdroje TWAIN (název a výrobce). Po dodání těchto voleb lze mikroovladač nainstalovat pomocí Správce zařízení[10] nebo k tomu určenou aplikací.

5.9 Server `scand`

Na straně operačního systému Linux je použit existující démon `saned` komunikující síťovým protokolem SANE. Pro Windows a protokol `TWAIN-NET` je vytvořen server nový. Zdrojové kódy aplikace, koncipované jako služby systému [31], jsou umístěny v adresáři `server-service`.

Hlavní implementační soubor `main.cpp` využívá třídy `TwProtocol::Async::Server` pro zajištění naslouchání na portu protokolu TCP. Nově vytvořená třída `Conn` (dědí třídu `TwProtocol::Async::ClientConnection`) zajišťuje odpovědi na dotazy TWAIN. Tato část serveru pouze obstarává inicializaci spojení, autentizaci a získání seznamu připojených zařízení. Při požadavku na otevření datového zdroje dochází k vytvoření nového procesu dle typu a architektury zařízení, který již příchozí požadavky TWAIN zpracovává. Rozdělení činnosti do dvou oddělených procesů je možné díky mechanismu popsanému v kapitole 5.5. Samotné vytvoření nového procesu obstarává funkce `Process::start()` z kapitoly 5.1.3, které je předán funkční objekt, jež po ukončení procesu sezení navrací starost o zpracování požadavků zpět do procesu služby.

Konfigurace serverové služby je obsažena ve třídě `Storage` (`config.hpp`, `config.cpp`) a pomocných třídách. Při každém požadavku na otevření zařízení je v odpovídající instanci třídy `Device` inkrementován čítač spojení a při zavření zařízení je snížen. Tyto operace jsou zajištěny v řadě konstruktorem třídy `Connection` a jejím destruktorem; instance je držena, dokud není zařízení uzavřeno. Zasláním uživatelského požadavku číslo 128 službě serveru dochází k opětovnému načtení a aplikaci nastavení ze souboru.

[10] Nedílná součást systémů Windows.

5.9.1 Sezení TWAIN

Implementace sezení typu TWAIN je díky frameworku TWAIN++ (5.4) jednoduchá. Každá operace ve třídě `Session` (`twain-session/main.cpp`), která je protipólem `Conn` z předešlé kapitoly, pouze volá odpovídající funkce TWAIN++. Inicializace spojení, autentizace a získání seznamu zařízení nejsou podporovány, resp. při jejich volání je navrácena chybová hodnota, ale nedochází ke změnám spojení.

5.9.2 Sezení WIA

Protipólem `Conn` je v případě sezení WIA třída `Session` definovaná v souboru `wia-session/main.cpp`. Základní struktura je shodná s implementací sezení TWAIN, avšak zde dochází k překladu mezi dvěma rozdílnými rozhraními. Sezení WIA je implementováno jako klient rozhraní WIA ve verzi 1. Tabulka 10 v příloze 5 uvádí podporované operace TWAIN++ a jejich mapování na funkce nebo atributy WIA.

Získání obrazových dat je realizováno při prvním požadavku připojeného klienta metodou `IWiaDataTransfer::idtGetData()`, jejíž výstup je uložen do dočasného souboru ve formátu BMP. Ke čtení dat je použit objekt `Gdiplus::Bitmap` [29] a jeho metoda `LockBits()`. Před každým čtením souboru je nastavena následující nepřečtená oblast obrázku. Přenos probíhá sémanticky zvrchu dolů. Barevný obrázek je ukládán ve formátu BGR, proto je po uložení do výstupní paměti provedena konverze BGR na RGB (spočívá pouze v záměně odpovídajících bajtů). Podporovány jsou pouze 24bitové barevné obrázky, 8bitové ve stupních šedi a 1bitové černobílé.

Pro mírné ulehčení práce s modelem COM a rozhraním WIA byly vytvořeny podpůrné třídy. Třída `ComScoped` (`comscoped.hpp`) obaluje objekty COM a zajišťuje volání metody `IUnknown::Release()` při destrukci instance. Vzhledem k implementaci jako šablony jazyka C++, je možné použít přetížený operátor šipky k přímému volání metod obaleného objektu. Třídy `PropVariant` (`propvariant.hpp`) a `StgMedium` (`stdmedium.hpp`) obalují struktury `PROPVARIANT` (zde pole) a `STGMEDIUM` a v destruktoru je uvolňují odpovídající funkcí.

5.10 Konfigurační aplikace

Pro uživatelský komfort jsou na operačním systému Windows dodány konfigurační aplikace klienta a serveru. Projekt je plně funkční i bez těchto aplikací, proto budou podrobněji uvedeny pouze zajímavější části jejich implementace. Uživatelské rozhraní vychází z [19] a je vytvořeno pomocí frameworku Qt [16].

5.10.1 Klient

Klientská konfigurační aplikace (podprojekt `config/client`) sestává ze tří grafických částí: hlavního okna, serverového dialogu a dialogu zařízení. Hlavní okno zobrazuje nastavené servery a zařízení

a jejich vazby. Pomocí serverového dialogu lze přidat nový server či změnit nastavení a zpřístupněná zařízení stávajícího serveru. Dialog zařízení zajišťuje změnu lokálního jména a výrobce datového zdroje a vybrané implementace grafického rozhraní a nativní přenosové metody. Povolení nového zařízení je provázeno vytvořením nového lokálního datového zdroje TWAIN, zařízení WIA a aktualizací souboru s nastavením. Změna zařízení vyžaduje, mimo zápisu do souboru, reinstalaci odpovídajícího ovladače WIA s aplikovanými úpravami. Datový zdroj TWAIN není třeba měnit, dostačuje změna souboru s nastavením.

Instalace, odstranění a získání dostupných zařízení WIA je řešeno funkcemi aplikačního rozhraní SetupAPI [32]. Při instalaci je do souboru INF dodána sekce se jménem zařízení, seznamem dostupných rozlišení a identifikace odpovídajícího datového zdroje TWAIN (název a výrobce). Práce s ovladači vyžaduje, aby architektura aplikace byla totožná s architekturou systému, tedy na 64bitovém systému musí být použita 64bitová aplikace, nikoliv 32bitová v emulační vrstvě. Při spuštění je tedy provedena pomocí funkce `Utils::isWow64` kontrola architektury a případně spuštěna odpovídající verze konfigurační aplikace.

Třída `NetClient` (client.hpp, client.cpp) zajišťuje abstrakci síťových protokolů `TWAIN-NET` (5.5) a SANE (5.3). Jedná se o jednotný systém připojení k serveru, autentizace a získání seznamu dostupných zařízení. Také je získán seznam možných rozlišení.

5.10.2 Server

Hlavní okno serverové konfigurační aplikace (podprojekt `config/server/utility`) zobrazuje seznam zpřístupněných zařízení a uživatelů a umožňuje měnit nastavení služby. Uživatelský dialog je použit ke změně jména a hesla stávajícího uživatele a k vytvoření nového. Dialog seznamu zařízení ukazuje veškerá dostupná zařízení TWAIN i WIA, uživatel je zde může povolit nebo zakázat. Dialog zařízení slouží k zobrazení informací o zařízení a limitaci počtu připojení.

Třídy `Manager` a `Service` (`service.hpp`, `service.cpp`) abstrahují funkce pro práci se službami systému Windows [31] do jednoduchého rozhraní. První jmenovaná třída umožňuje vytvoření nové služby a její otevření. Instance `Service` následně tuto službu ovládá, tedy jde o její spuštění, zastavení, získání stavu a odstranění. Také lze službě zaslat uživatelský dotaz, který je zde využit k upozornění služby serveru na změnu konfigurace.

K zajištění spolehlivosti aplikace a z důvodu přítomnosti 32bitového i 64bitového rozhraní TWAIN na 64bitovém systému Windows, bylo samotné získání dostupných zařízení přesunuto do zvláštních aplikací (podprojekty `config/twain-finder` a `config/wia-finder`). Obě aplikace zapisují seznam svých zařízení na standardní výstup, přičemž nainstalovaná zařízení tohoto projektu jsou ignorována. Neboť otevření a práce se zařízením může vyvolat neočekávaný zápis do standardního výstupu, a tedy špatný formát dat, je použit objekt `Utils::StdoutSuppress` k jeho dočasnému zakázání. K povolení vždy dojde těsně před výstupem položky seznamu.

6 Testování

Pro účely otestování projektu byla použita multifunkční zařízení Canon MF3220 a Brother DCP-9020CDW, virtuální zařízení TWAIN2 Software Scanner [33] (použita nejnovější verze 2.1.4 z repositáře, nikoliv z předpřipravených archivů) pro rozhraní TWAIN, pod rozhraním SANE virtuální testovací backend `test`, který je součástí distribuce. Hostovacím systémem datových zdrojů TWAIN a zařízení WIA byl systém Windows 7 v 32bitové verzi, backendy a služby SANE byly spouštěny na 32bitové linuxové distribuci Ubuntu 14.04 LTS.

Vytvořený síťový datový zdroj TWAIN byl testován nástrojem Inspector TWAIN [34] a ukázkovou aplikací z [33]. V případě zařízení WIA byl pouze proveden manuální sken v programu Malování (`mspaint.exe`). Stejným způsobem byl otestován backend SANE programem XSane 0.999 [35]. Pro porovnání výsledků bylo nejprve otestováno samotné zařízení. Následující kapitoly popisují průběh testování podle zpřístupňovaného zařízení a skenovacího rozhraní, ke kterému bylo připojeno.

6.1 TWAIN

6.1.1 Canon MF3220

V průběhu testování byla neočekávaně zobrazena dialogová okna datového zdroje, která bylo nutno manuálně uzavřít, jinak nebylo možno pokračovat. Stejný případ nastává při zastavení přenosu uživatelem. Testování počtu přenosů vyústilo v nekonečnou skenovací smyčku, uživatelské zastavení mělo za následek tzv. „zamrznutí“ programu, který tak musel být násilně ukončen. Při přeskočení tohoto dílčího testu docházelo ke stejné situaci při testování nativní metody. Tento datový zdroj je chybný, testování funkčnosti projektu nad ním nemá smysl.

Jednoduché získání obrazu pomocí ukázkové aplikace proběhlo korektně nad samotným zařízením i při přístupu přes síťový datový zdroj. Backend SANE ani zařízení nad tímto skenerem nebyly vyzkoušeny.

6.1.2 Brother DCP-9020CDW

Test počtu přenosů má za následek přerušení testování s chybou indikující nepřenesení žádných dat. Rychlý přenos velkého množství obrázků nativní metodou zapříčiní zamrznutí aplikace. Jednotlivé testy přenosových metod bez i se zobrazeným GUI jsou ukončeny chybou, není přenesen žádný obraz. Datový zdroj pro toto zařízení je také chybný, a nebude tedy použit v dalším testování projektu.

Pokus o přenos obrazu ukázkovou aplikací byl úspěšný pouze při přímém připojení. Síťová komunikace umožňuje nastavení vzdáleného zdroje, avšak pokus o získání obrazu selhal.

6.1.3 TWAIN2 Software Scanner

Byla odhalena chyba při přenosu nedefinovaného počtu obrázků, kdy byl přenesen pouze jeden, ale očekávány dva. Ani oficiální ukázkový datový zdroj tedy není naprosto korektně implementován, avšak nejde o zásadní chybu, která by znemožňovala jeho použití.

Výsledek testování při využití síťového datového zdroje je naprosto totožný, pouze nebyly použity nepodporované přenosové metody. Zařízení WIA umožňuje správně nastavit model barev i výseč obrázku. První získání obrazu však vrací levou horní výseč, přičemž ignoruje nastavení rozlišení. Veškeré další přenosy probíhají správně. V případě SANE jsou veškeré barevné obrázky i obrázky v odstínech šedi přenášeny bez problémů. Přijaté černobílé obrázky obsahují invertované barvy (na místě bílé černá a naopak), při rozlišení 50 DPI je přenos ukončen bez přijetí dat.

6.2 WIA

6.2.1 Canon MF3220

Testy aplikace Inspector TWAIN dopadly uspokojivě. Ovladač zařízení neupravuje možné barevné hloubky podle vybraného barevného modelu, proto dochází ke kombinacím nepodporovaným zařízením i překladovou vrstvou. Při získávání barevného obrázku s vyšším rozlišením došlo na straně klienta k vypršení časovače a odpojení od serveru, nastavení vyšší hodnoty problém vyřešilo. Překlad na rozhraní WIA (tedy „WIA na TWAIN na WIA“) pracuje korektně s výjimkou nastavení výseče obrazu, vždy je přenesena levá horní část obrazu. Backend SANE vykazuje stejný problém. Ukázková aplikace TWAIN přenáší obraz celý.

6.2.2 Brother DCP-9020CDW

I zde dochází k přeskočení nepodporovaných kombinací barevné hloubky a modelu. Všechna tři přístupová rozhraní korektně přenáší celý obraz, nedochází k přenosu pouze jeho části.

6.3 SANE

6.3.1 Canon MF3220

Testovací sada nalezla pouze podružné chyby: lze zadat rozlišení s hodnotou nula a přenosy v rychlém sledu za sebou selžou. Obě chyby jsou způsobeny na straně backendu nehlídáním vstupu a okamžitým návratem ze čtecí funkce `sane_read()` namísto čekání na data. Ukázková aplikace TWAIN fungovala správně. Při prvním přístupu pomocí rozhraní WIA došlo opět k získání pouze části obrazu, následující

přenosy proběhly celé. Bylo možno získat pouze barevný obrázek, neboť překladová vrstva nepodporuje získání a nastavení barevného modelu.

6.3.2 Brother DCP-9020CDW

Backend či přímo zařízení reaguje velmi pomalu i při přímém připojení, docházelo k nemožnosti připojit se, kdy pomohl pouze restart skeneru. Projevila se stejná chybná možnost nastavit nulové rozlišení jako v předchozím případě. Aplikace TWAIN mohla získat obraz v nižších rozlišeních, vyšší nebylo možné přenést z důvodu nedostatku paměti v démonu `saned`. Překladová vrstva WIA nebyla schopna doručit obraz aplikaci, zařízení čekalo na odběr dat.

6.3.3 Backend `test`

Veškeré testy TWAIN dopadly úspěšně. V případě WIA opět nebylo možné získat obrazová data, byla pouze zobrazena chybová hláška.

6.4 Vyhodnocení testů

Implementace síťových protokolů TWAIN i SANE jsou bez zjevných chyb, nebyl zjištěn přístup mimo vyhrazenou paměť. U všech skenovacích rozhraní závisí možnost sdílení zařízení především na kvalitě jeho ovladače (datového zdroje, backendu). Datové zdroje obou fyzických skenerů vykazovaly závažné chyby, bylo možné zpřístupnit je Canon MF3320, a to pouze v omezené míře. Překladová vrstva SANE přítomná v datovém zdroji nevykazuje zásadní chyby. Před případným ostrým nasazením bude nutné prozkoumat možnosti detekce chyb a zotavení z nich. Také bude vhodné rozšířit a dále otestovat překlad voleb SANE.

Testy rozhraní WIA na straně serveru dopadly úspěšně s výhradami. Potenciální problém spočívá v přenosu obrazu s vyšším rozlišením, neboť může docházet k vypršení časovače na straně klienta, server nejprve získá celý obraz, a až poté jej zasílá. Nastavení delší prodlevy vede k úspěšnému přenosu. Pro další nasazení bude vhodné přepracovat aplikaci sezení tak, aby zasílala data průběžně.

Mikroovladač WIA vykazuje značné chyby, dochází k přenosům pouhé části obrazu nebo končí přímo s chybou. Pravděpodobnou příčinou je využití modelu mikroovladače samotného namísto plné implementace miniovladače. Nadřazený miniovladač příliš omezuje možnosti nastavení. Pokud vzdálené zařízení neposkytuje odpovídající volby v požadovaném čase, dochází k chybám. Řešením je implementace plného miniovladače s pracovním postupem podobným rozhraní TWAIN.

Backend SANE funguje správně s výjimkou přenosu černobílého obrazu, jehož barvy jsou interpretovány inverzně. V jednom případě došlo k opakovanému přenosu pouhé části obrázku, pro potvrzení původu chyby budou nutné testy na dalších zařízeních.

7 Limity projektu

7.1 TWAIN

Hlavním omezujícím faktorem části projektu, která pracuje s rozhraním TWAIN, je navržený síťový protokol. Je podporována pouze omezená množina kontrolních a obrazových příkazů. U neimplementovaných příkazů, které má smysl přenášet, záleží možnost jejich podpory na významu položek v odpovídajících strukturách, především lze-li získat jejich přesnou velikost. Podporu operací nad souborovým systémem lze realizovat přenosem obsahu souboru a změnou cesty k němu. Jedinou výjimku tvoří události zařízení. Přenos samotné datové struktury je triviální, současný protokol však neumožňuje serveru zasílat zprávy, které nejsou odpovědí na dotaz klienta.

Způsob implementace také může vést k nepodpoře částí rozhraní TWAIN. Nativní přenosy nejsou přítomny přímo v síťovém protokolu, lze je však realizovat na straně klienta pomocí paměťové přenosové metody. Tato metoda však umožňuje přenos dat v různých formách, rozdílné barevné hloubce i modelu, s kompresí či bez ní atp. Není reálně možné, aby jediná implementace obsáhla všechny možnosti. Překlad metod však byl přesunut do oddělené knihovny a poskytnut hlavičkový soubor s definicí rozhraní, lze tedy vytvořit implementaci pro datové zdroje, které využívají formát nepodporovaný výchozí knihovnou.

Mírný problém nastává i u rozšířených informací o obraze. Z typu `Handle` nelze jednoduše odvodit velikost dat. K odvození velikosti je nutné znát význam odpovídajícího atributu. Atributy s tímto typem, které byly specifikovány po vytvoření implementace, tedy nemohou být podporovány. Implementace serveru v tomto projektu atributy s typem `Handle` nepřenáší, namísto toho je nastaven příznak jeho nepodpory.

7.2 WIA

Odhlédneme-li od přenosového protokolu, je serverová část, tedy překlad z rozhraní WIA na TWAIN (TWAIN++), limitována pouze kvalitou implementace. Volby WIA nemohou být automaticky mapovány na volby TWAIN, je třeba pro každou z nich konzultovat specifikace obou rozhraní. Získání obrazových dat také vyžaduje překlad. Implementace v tomto projektu podporuje pouze základní operace tak, aby kontrola specifikace podle aplikace Inspector TWAIN proběhla úspěšně. Podpora vstupních formátů zde závisí na knihovně GDI+, jako výstup jsou implementovány pouze barevné obrázky s 24bitovou barevnou hloubkou, šedé 8bitové a 1bitové černé obrázky.

Podpora operací v mikroovladači WIA je ovlivněna již jeho samotnou omezenou architekturou. Také pořadí operací a způsob jejich vyhodnocení jsou zásadně odlišné od rozhraní TWAIN. Není zde přítomen prvek identifikující jednotlivá sezení, proto každá operace musí otevřít odpovídající datový

zdroj a nastavit jej, až nyní je operace provedena (a zdroj uzavřen). Tento způsob fungování může vyústit v rozdílech nastavení v částech pracujících s obě rozhraními. Jak bylo zjištěno v kapitole 6, dojde často až k nemožnosti získat obrazová data.

7.3 SANE

Síťový backend provádí překlad rozhraní TWAIN na SANE, má výhradní kontrolu nad nastavením vzdáleného datového zdroje. Základní množina operací je plně pokryta povinně dostupnými protějšky rozhraní TWAIN. Množina obrazových formátů v SANE je podstatně menší, datový zdroj, který nepodporuje odpovídající typy, nemůže být zpřístupněn bez překladu dat. Implementace v tomto projektu překlad dat neprovádí. V případě voleb je situace obdobná jako u rozhraní WIA, překlad musí být implementován manuálně pro každou volbu zvlášť. Rozhraní SANE ponechává veškerou funkcionalitu voleb na tvůrci backendu (kromě několika tzv. „well-known“, tedy „známé“). Tento projekt implementuje pouze známé volby a volby pro nastavení rozlišení v obou směrech zvlášť a barevné hloubky a modelu obrazu.

I na straně datového zdroje naráží překlad z rozhraní SANE na velmi malou množinu známých voleb, které navíc nemusí být přítomny. Pro možnost nastavení dalších voleb bude nutno manuálně prozkoumat volby existujících backendů a vytvořit překladová pravidla. Dále implementace nepodporuje přenos obrazových dat v oddělených barevných rámcích.

8 Možnosti rozšíření, úpravy

8.1 Podpora více operací

Protokol TWAIN-NET v tuto chvíli umožňuje vzdálené volání pouhé části všech možných operací rozhraní TWAIN. Případná rozšíření projektu mohou jednoduše přidat další operace včetně přenosových metod obrazu a popř. i zvuku. Podpora událostí zařízení bude vyžadovat změnu v síťovém protokolu tak, aby bylo možné zasílat zprávy klientovi mimo pořadí. Při implementaci rozšíření bude třeba na klientovi zajistit změnu případných cest k souborům na cesty na lokálním systému, před tím však bude muset dojít k jejich přenosu.

Backend pro rozhraní SANE a překladová vrstva WIA na serveru zpřístupňují pouhé minimum voleb/atributů, bude vhodné přidat další. Veškerá práce s tím spojená ale nemůže být automatizována, pro každou volbu či atribut musí dojít ke konzultaci specifikací zdrojového i cílového rozhraní.

8.2 Miniovladač WIA

V současné době je klientská část rozhraní WIA implementována jako mikroovladač. Jeho testování odhalilo několik nedostatků, které vyústily až v nemožnost získat obraz. Tento problém je s velkou pravděpodobností způsoben příliš jednoduchou architekturou mikroovladače, která ani neumožňuje spravovat sezení. Při případném dalším rozvoji projektu bude vhodné prozkoumat zevrubněji možnosti plného miniovladače a provést jeho implementaci.

8.3 Bezpečnost

Tento bod je již přítomen v [19], tedy lepší možnosti správy uživatelských účtů, možnosti účtování a volitelné šifrování síťového protokolu. V této práci jsme se soustředili na multiplatformnost řešení, implementace kvalitnějších bezpečnostních prvků bude realizována v případných dalších rozšířeních.

8.4 Robustnost

Testování datových zdrojů pro fyzická zařízení odhalilo velké množství chyb a odchylek od definovaného standardu. Při pokusech o vzdálený přístup byly přenosy obrazu předčasně ukončovány a docházelo k nekonečným smyčkám, popř. zobrazení dialogových oken, která neumožňovala pokračování programu bez uživatelského vstupu. Robustní program by takovým situacím měl předcházet, detekovat je a zotavit se z nich.

9 Závěr

Hlavní důraz v této práci jsem kladl na podporu tří skenovacích rozhraní na dvou rozdílných platformách. Systém umožňuje skenovat pomocí rozhraní TWAIN, WIA (Windows) a SANE (Linux). Výsledný produkt jsem navrhnul a implementoval tak, aby bylo možné získávat obrázky ze skenerů připojených k libovolnému ze zmíněných rozhraní pomocí aplikací, které využívají libovolné (i jiné) skenovací rozhraní.

Pro účely síťové komunikace jsem navrhnul nový aplikační protokol. Vycházel jsem ze zkušeností při návrhu původního protokolu, především jsem zjednodušil možnosti příjmu zpráv, nyní je možné celou zprávu načíst do paměti bez jejího zpracování.

Jako přenosové rozhraní jsem zvolil TWAIN, u ostatních (tedy WIA a SANE) probíhá překlad operací pomocí vytvořených překladových vrstev. Pouze v případě připojení k serveru, který komunikuje síťovým protokolem SANE, je použit právě protokol SANE a operace rozhraní TWAIN jsou do něj překládány.

Datový zdroj rozhraní TWAIN jsem nově implementoval za využití frameworku TWAIN++, který jsem k tomu účelu vytvořil. Účelem tohoto frameworku je podstatné zjednodušení vytváření aplikací i datových zdrojů a lepší práce s datovými typy. Každý datový zdroj musí obsahovat grafické uživatelské rozhraní, zde jsem jej přesunul do externí knihovny, lze tak jednoduše implementovat vlastní podle požadavků zpřístupněného zařízení. Původní práce přenáší obrazová data ve formátu závislém na původní platformě, nyní jsem vytvoření takových dat pomocí jiné přenosové metody umístil do klientské části. Navíc jsem potřebný kód zapsal do dynamické knihovny, tím jsem umožnil vytvoření vlastních implementací, které mohou podporovat další formáty vstupních i výstupních dat.

Implementace klientské část rozhraní SANE na linuxových operačních systémech, tzv. backend, je pouze minimální. Specifikace definuje pouhých několik voleb nastavení, které i tak nejsou povinné. Pro zachování celistvosti bude v případě přidání podpory pro větší množství voleb nutno prozkoumat existující backendy.

Ovladač zařízení WIA jsem navrhnul jako pouhou překladovou vrstvu nad datovým zdrojem. Při samotné implementaci jsem zvolil jednodušší architekturu mikroovladače namísto miniovladače. Toto rozhodnutí se při testech ukázalo jako nešťastné. Kvůli přílišné odlišnosti stylu práce a chybějící správě sezení docházelo až k nemožnosti přenést data.

Práci jsem implementoval v programovacím jazyce C++. Využil jsem knihovny Boost a pro grafická rozhraní framework Qt. Vytvořil jsem množství podpůrných knihoven, které budou mít uplatnění i mimo tuto práci. Především jde o již zmíněný framework TWAIN++, pro který plánuji další vývoj jako samostatný projekt.

Výstup této práce má celou řadu využití. Zjevné je použití ke vzdálenému skenování, kdy není nutné být fyzicky přítomen na počítači, k němuž je zařízení připojeno. Díky podpoře více rozhraní

a platforem lze jednoduše vyřešit problém chybějících ovladačů, skener je připojen k podporovanému rozhraní a pomocí této práce zpřístupněn v rozhraní požadovaném. Stejně tak je umožněno připojit 32bitovou aplikaci k 64bitovému ovladači zařízení a naopak.

V kapitole 8 jsem uvedl náměty na další rozšíření či úpravy tohoto projektu. Jde o již zmíněné použití plného miniovladače namísto omezeného mikroovladače v rozhraní WIA. V případě rozhraní TWAIN a SANE uvádím podporu většího množství operací a voleb nastavení.

Literatura

1. TWAIN WORKING GROUP. *TWAIN Specification: Version 2.3* [online]. 2013 [cit. 2014-11-11]. Dostupné z: http://www.twain.org/docs/530fe0da85f7511c510004ff/TWAIN-2.3-Spec.pdf
2. MICROSOFT CORPORATION. *Windows Image Acquisition (WIA)* [online]. 2014 [cit. 2014-11-11]. Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/ms630368%28v=vs.85%29.aspx
3. SANE PROJECT. *SANE Standard Version 1.04* [online]. 2006 [cit. 2014-11-11]. Dostupné z: http://www.sane-project.org/html/
4. *ISO/IEC 14882:2011*. ISO, 2011. Dostupné také z: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=60&ics3=&csnumber=50372
5. STROUSTRUP, B. *The C++ programming language*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 978-0-321-56384-2.
6. RFC 791. *Internet Protocol*. IETF, 1981. Dostupné také z: http://tools.ietf.org/html/rfc791
7. RFC 2460. *Internet Protocol, Version 6 (IPv6) Specification*. IETF, 1998. Dostupné také z: http://tools.ietf.org/html/rfc2460
8. RFC 793. *Transmission Control Protocol*. IETF, 1981. Dostupné také z: http://tools.ietf.org/html/rfc793
9. RFC 768. *User Datagram Protocol*. IETF, 1980. Dostupné také z: http://tools.ietf.org/html/rfc768
10. *Boost C++ Libraries* [online]. 2014, verze 1.57.0 [cit. 2014-11-11]. Dostupné z: http://www.boost.org/
11. Boost Software License [online]. 2003 [cit. 2014-11-11]. Dostupné z: http://www.boost.org/LICENSE_1_0.txt
12. KOHLHOFF, C. *Boost.Asio - 1.57.0* [online]. 2014 [cit. 2014-11-11]. Dostupné z: http://www.boost.org/doc/libs/1_57_0/doc/html/boost_asio.html
13. HENNEY, K. A. NASONOV a A. POLUKHIN. *Boost.Lexical_Cast* [online]. 2000-2014, verze 2014 [cit. 2014-11-11]. Dostupné z: http://www.boost.org/doc/libs/1_57_0/doc/html/boost_lexical_cast.html
14. KALICINSKI, M. *Chapter 22. Boost.PropertyTree - 1.57.0* [online]. 2008 [cit. 2015-05-07]. Dostupné z: http://www.boost.org/doc/libs/1_57_0/doc/html/property_tree.html
15. THE OPENSSL PROJECT. *OpenSSL* [online]. 1999-2014 [cit. 2014-11-11]. Dostupné z: https://www.openssl.org/
16. DIGIA. *Qt Project* [online]. 2014 [cit. 2014-11-11]. Dostupné z: http://qt-project.org/

17. MICROSOFT CORPORATION. *Imaging Devices (Scanner)* [online]. 2014 [cit. 2014-12-11]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff546215%28v=vs.85%29.aspx
18. MICROSOFT CORPORATION. *COM: Component Object Model Technologies* [online]. 2014 [cit. 2014-12-11]. Dostupné z: http://www.microsoft.com/com/default.mspx
19. RICHTER, M. *Síťové rozšíření systému TWAIN*. Brno: 2011. Dostupné také z: https://wis.fit.vutbr.cz/FIT/st/rp.php/rp/2011/BP/14080_f.pdf
20. SHS. *Secure Hash Standard*. FIPS, 2012. Dostupné také z: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
21. GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Massachusetts: Addison-Wesley, 1995. ISBN 0-201-63361-2.
22. NIST. *Recommendation for Block Cipher Modes of Operation* [online]. 2001 [cit. 2015-04-29]. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
23. RFC 2315. *PKCS #7: Cryptographic Message Syntax*. IETF, 1998. Dostupné také z: https://tools.ietf.org/html/rfc2315
24. RFC 4648. *The Base16, Base32, and Base64 Data Encodings*. IETF, 2006. Dostupné také z: http://tools.ietf.org/html/rfc4648
25. NYFFENEGGER, R. René Nyffenegger's collection of things on the web. In: *Encoding and decoding base64 with C++* [online]. 2004-008 [cit. 2015-04-29]. Dostupné z: http://www.adp-gmbh.ch/cpp/common/base64.html
26. MACIEIRA, T. Thiago Macieira's blog. In: *Initialising an array with C++0x using constexpr and variadic templates* [online]. 2011 [cit. 2015-04-29]. Dostupné z: http://www.macieira.org/blog/2011/07/initialising-an-array-with-cx0x-using-constexpr-and-variadic-templates/
27. MICROSOFT. Microsoft Developer Network. In: *IsWow64Process function* [online]. © 2015 [cit. 2015-04-30]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms684139%28VS.85%29.aspx
28. Microsoft Developer Network. *BITMAPINFO structure* [online]. © 2015 [cit. 2015-05-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd183375%28v=vs.85%29.aspx
29. Microsoft Developer Network. *GDI+* [online]. 2012 [cit. 2015-05-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms533798%28v=vs.85%29.aspx
30. Microsoft Developer Network. *Overview of INF Files* [online]. © 2015 [cit. 2015-05-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff549520.aspx
31. Windows Dev Center. *Services* [online]. © 2015 [cit. 2015-05-11]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685141%28v=vs.85%29.aspx

32. Hardware Dev Center. *SetupAPI Reference* [online]. © 2015 [cit. 2015-05-12]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/hardware/ff550897%28v=vs.85%29.aspx
33. TWAIN sample Data Source and Application [online]. 2013 [cit. 2015-05-14]. Dostupné z: http://sourceforge.net/projects/twain-samples/
34. Inspector TWAIN [online]. 2008 [cit. 2015-05-14]. Dostupné z: http:// www.inspectortwain.com/
35. RAUCH, O. XSane - graphical scanning frontend [online]. 2013 [cit. 2015-05-14]. Dostupné z: http://www.xsane.org/

Seznam příloh

Příloha 1

Datové typy ve frameworku TWAIN++

Název, soubor	Prefix TWAIN	Poznámka
Alarm alarm.hpp	TWAL_	Volba typu CapType::Alarms (CAP_ALARMS).
Detail::ArgTypes argumenttype.hpp	DAT_	Argument vstupní funkce TWAIN specifikující typ dat.
AutoSize autosize.hpp	TWAS_	Volba typu CapType::IAutoSize (ICAP_AUTOSIZE).
BarCodeType barcodetype.hpp	TWBT_	Volba typu CapType::ISupportedBarcodeTypes (ICAP_SUPPORTEDBARCODETYPES).
BitDepthReduction bitdepthreduction.hpp	TWBR_	Volba typu CapType::BitDepthReduction (ICAP_BITDEPTHREDUCTION).
BitOrder bitorder.hpp	TWBO_	Volba typu CapType::IBitOrder (ICAP_BITORDER).
CameraSide cameraside.hpp	TWCS_	Volba typu CapType::CameraSide (CAP_CAMERASIDE).
ClearBuffers clearbuffers.hpp	TWCB_	Volba typu CapType::ClearBuffers (CAP_CLEARBUFFERS).
ColorFormat (PlanarChunky) colorformat.hpp	TWPC_	Volba typu CapType::IPlanarChunky (ICAP_PLANARCHUNKY).
Compression compression.hpp	TWCP_	Volba typu CapType::ICompression (ICAP_COMPRESSION), užito v ImageInfo (TW_IMAGEINFO), ImageMemXfer a ImageMemFileXfer (TW_IMAGEMEXFER).
Country country.hpp	TWCY_	Země původu, použito ve Version (TW_VERSION).

Název, soubor	Prefix TWAIN	Poznámka
DataGroup (DG) datagroup.hpp	DG_	Argument vstupní funkce TWAIN specifikující kategorii operace. Použito i v Identity (TW_IDENTITY) a jako datový argument vstupní funkce. Lze využít jako příznaky („flags“) díky definovaným logickým operátorům (negace, logický součin, logický součet).
DiscardBlankPages discardblankpages.hpp	TWBP_	Volba typu CapType::IAutoDiscardBlankPages (ICAP_AUTODISCARDBLANKPAGES). Jde o speciální hodnoty, vlastní konkrétní hodnoty lze získat užitím funkce discardBytes().
DoubleFeedDetection doublefeeddetection.hpp	TWDF_	Volba typu CapType::DoubleFeedDetection (CAP_DOUBLEFEEDDETECTION).
DoubleFeedResponse doublefeedresponse.hpp	TWDP_	Volba typu CapType::DoubleFeedDetectionResponse (CAP_DOUBLEFEEDDETECTIONRESPONSE).
DoubleFeedSensitivity doublefeedsensitivity.hpp	TWUS_	Volba typu CapType::DoubleFeedDetectionSensitivity (CAP_DOUBLEFEEDDETECTIONSENSITIVITY).
Duplex duplex.hpp	TWDX_	Volba typu CapType::Duplex (CAP_DUPLEX).
FeederAlignment feederalignment.hpp	TWFA_	Volba typu CapType::FeederAlignment (CAP_FEEDERALIGNMENT).
FeederOrder feederorder.hpp	TWFO_	Volba typu CapType::FeederOrder (CAP_FEEDERORDER).
FeederPocket feederpocket.hpp	TWFP_	Volba typu CapType::FeederPocket (CAP_FEEDERPOCKET).
FeederType feedertype.hpp	TWFE_	Volba typu CapType::IFeederType (ICAP_FEEDERTYPE).
FilmType filmtype.hpp	TWFM_	Volba typu CapType::IFilmType (ICAP_FILMTYPE).
Filter filter.hpp	TWFT_	Volba typu CapType::IFilter (ICAP_FILTER).

Název, soubor	Prefix TWAIN	Poznámka
Flash flash.hpp	TWFL_	Volba typu CapType::IFlashUsed2 (ICAP_FLASHUSED2), užito v DeviceEvent (TW_DEVICEEVENT).
FlipRotation fliprotation.hpp	TWFR_	Volba typu CapType::IFlipRotation (ICAP_FLIPROTATION).
FontStyle fontstyle.hpp	TWPF_	Volba typu CapType::PrinterFontStyle (CAP_PRINTERFONTSTYLE).
IccProfile iccprofile.hpp	TWIC_	Volba typu CapType::IIccProfile (ICAP_ICCPROFILE).
ImageFileFormat imagefileformat.hpp	TWFF_	Volba typu CapType::IImageFileFormat (ICAP_IMAGEFILEFORMAT).
ImageFilter imagefilter.hpp	TWIF_	Volba typu CapType::IImageFilter (ICAP_IMAGEFILTER).
ImageMerge imagemerge.hpp	TWIM_	Volba typu CapType::IImageMerge (ICAP_IMAGEMERGE).
IndexTrigger indextrigger.hpp	TWCT_	Volba typu CapType::PrinterIndexTrigger (CAP_PRINTERINDEXTRIGGER).
IndicatorsMode indicatorsmode.hpp	TWCI_	Volba typu CapType::IndicatorsMode (CAP_INDICATORSMODE).
JobControl jobcontrol.hpp	TWJC_	Volba typu CapType::JobControl (CAP_JOBCONTROL).
JpegSubSampling jpegsubsampling.hpp	TWJS_	Volba typu CapType::IJpegSubSampling (ICAP_JPEGSUBSAMPLING).
Language language.hpp	TWLG_	Volba typu CapType::Language (CAP_LANGUAGE), užito ve Version (TW_VERSION).
LightPath lightpath.hpp	TWLP_	Volba typu CapType::ILightPath (ICAP_LIGHTPATH).
LightSource lightsource.hpp	TWLS_	Volba typu CapType::ILightSource (ICAP_LIGHTSOURCE).
Mirror mirror.hpp	TWMR_	Volba typu CapType::IMirror (ICAP_MIRROR).

Název, soubor	Prefix TWAIN	Poznámka
Msg msg.hpp	MSG_	Argument vstupní funkce TWAIN specifikující typ akce, jež má být nad zadanými daty provedena.
MsgSupport msg.hpp	TWQC_	Příznaky podporovaných akcí (výčet Msg) nad konkrétní vybranou volbou.
NoiseFilter noisefilter.hpp	TWNF_	Volba typu CapType::INoiseFilter (ICAP_NOISEFILTER).
Orientation orientation.hpp	TWOR_	Volba typu CapType::IOrientation (ICAP_ORIENTATION).
OverScan overscan.hpp	TWOV_	Volba typu CapType::IOverscan (ICAP_OVERSCAN).
PaperHandling paperhandling.hpp	TWPH_	Volba typu CapType::PaperHandling (CAP_PAPERHANDLING).
PaperSize papersize.hpp	TWSS_	Volba typu CapType::ISupportedSizes (ICAP_SUPPORTEDSIZES).
PixelFlavor pixelflavor.hpp	TWPF_	Volby typu CapType::IPixelFlavor (ICAP_PIXELFLAVOR), CapType::IPixelFlavorCodes (ICAP_PIXELFLAVORCODES), užito v rozšířené obrazové informační struktuře typu InfoId::PixelFlavor (TWEI_PIXELFLAVOR).
PixelType pixeltype.hpp	TWPT_	Volba typu CapType::IPixelType (ICAP_PIXELTYPE), užito v ImageInfo (TW_IMAGEINFO).
PowerSupply powersupply.hpp	TWPS_	Volba typu CapType::PowerSupply (CAP_POWERSUPPLY), užito v DeviceEvent (TW_DEVICEEVENT).
Printer printer.hpp	TWPR_	Volba typu CapType::Printer (CAP_PRINTER).
PrinterMode printermode.hpp	TWPM_	Volba typu CapType::PrinterMode (CAP_PRINTERMODE).
SearchMode searchmode.hpp	TWBD_	Volba typu CapType::IBarCodeSearchMode (ICAP_BARCODESEARCHMODE).

Název, soubor	Prefix TWAIN	Poznámka
Segmented segmented.hpp	TWGS_	Volba typu CapType::Segmented (CAP_SEGMENTED).
Unit unit.hpp	TWUN_	Volba typu CapType::IUnits (ICAP_UNITS).
XferMech xfermech.hpp	TWSX_	Volba typu CapType::IXferMech (ICAP_XFERMECH), pouze hodnoty Native a File pro volbu typu CapType::AXferMech (ACAP_XFERMECH).
InfoId extimageinfo.hpp	TWEI_	Užito v Info, popř. InfoRaw (obojí obdoba TW_INFO).
BarCodeRotation extimageinfo.hpp	TWBCOR_	Info typu InfoId::BarCodeRotation (TWEI_BARCODEROTATION).
PageSide extimageinfo.hpp	TWCS_	Info typu InfoId::PageSide (TWEI_PAGESIDE). Oproti CameraSide není podporována hodnota Both.
DeskewStatus extimageinfo.hpp	TWDSK_	Info typu InfoId::DeskewStatus (TWEI_DESKEWSTATUS).
MagType extimageinfo.hpp	TWMD_	Info typu InfoId::MagType (TWEI_MAGTYPE).
PatchCode extimageinfo.hpp	TWPCH_	Info typu InfoId::PatchCode (TWEI_PATCHCODE).
ConType capability.hpp	TWON_	Typ kontejneru volby Capability (TW_CAPABILTY).
CapType capability.hpp	CAP_ ACAP_ ICAP_	Typ volby Capability (TW_CAPABILITY).
DeviceEvent::Type deviceevent.hpp	TWDE_	Typ události DeviceEvent (TW_DEVICEEVENT).
FileSystem::FileType filesystem.hpp	TWFY_	Typ uzlu či listu ve FileSystem (TW_FILESYSTEM).
Palette8::Type palette8.hpp	TWPA_	Typ obrazové palety Palette8 (TW_PALETTE8).
PassThrough::Direction passthrough.hpp	TWDR_	Směr dat v PassThrough (TW_PASSTHRU).
PendingXfers::JobPatch pendingxfers.hpp	TWEJ_	Dodatečné informace o přenosu dle nastavené volby CapType::JobControl (CAP_JOBCONTROL).

Tabulka 1: Výčtové typy TWAIN++

Název, soubor	Typ TWAIN	Poznámka
AudioInfo audioinfo.hpp	TW_AUDIOINFO	
Detail::CallBack callback.hpp	TW_CALLBACK	Interní použití.
Detail::CallBack2 callback.hpp	TW_CALLBACK2	Interní použití.
CieColor ciecolor.hpp	TW_CIECOLOR	Význam a velikost pole samples (TW_CIECOLOR.Samples) není z manuálu [1] zřejmý. Ve struktuře TW_CIECOLOR je velikost tohoto pole nastavena na jeden prvek, popis hovoří o šesti položkách A, B, C, L, M a N. Podpora je pouze experimentální.
CiePoint ciepoint.hpp	TW_CIEPOINT	Součást CieColor (TW_CIECOLOR).
TransformStage transformstage.hpp	TW_TRANSFORMSTAGE	Součást CieColor (TW_CIECOLOR).
DecodeFunction decodefunction.hpp	TW_DECODEFUNCTION	Součást TransformStage (TW_TRANSFORMSTAGE).
Palette8 palette8.hpp	TW_PALETTE8	Automaticky nastavuje indexy položek. V C++11 lze vytvořit inicializované instance při překladu díky Utils::ArrData.
Element8 element8.hpp	TW_ELEMENT8	Součást Palette8 (TW_PALETTE8), RgbResponse (TW_RGBRESPONSE) a GrayResponse (TW_GRAYRESPONSE).
DeviceEvent deviceevent.hpp	TW_DEVICEEVENT	Obsahuje statické metody vyžadující zadání pouze validních položek pro vybraný typ události. Ostatní položky jsou nastaveny na výchozí hodnoty.
Detail::EntryPoint entrypoint.hpp	TW_ENTRYPOINT	Interní použití.
Detail::Event event.hpp	TW_EVENT	Použito pouze na Windows a Mac OS před verzí X. Veřejné při implementaci datového zdroje, jinak pouze interní použití.

Název, soubor	Typ TWAIN	Poznámka
FileSystem filesystem.hpp	TW_FILESYSTEM	
Identity identity.hpp	TW_IDENTITY	Skrývá nastavení verze protokolu, ID instance a podporovaných kategorií operací. Lze vytvořit také použitím třídy ImageBuilder, která umožňuje nastavit libovolnou položku.
Version identity.hpp	TW_VERSION	Součást Identity (TW_IDENTITY).
ImageInfo imageinfo.hpp	TW_IMAGEINFO	V C++11 lze vytvořit inicializované instance při překladu díky Utils::ArrFlatImpl.
ImageLayout imagelayout.hpp	TW_IMAGELAYOUT	Zadány výchozí hodnoty číselných položek pro aplikace a datové zdroje, které je nepoužívají.
PassThrough passthrough.hpp	TW_PASSTHRU	
PendingXfers pendingxfers.hpp	TW_PENDINGXFERS	
SetupFileXfer setupfilexfer.hpp	TW_SETUPFILEXFER	Na Windows a Linuxu skrývá položku volRefNum (TW_SETUPFILEXFER.VRefNum), která zde nemá význam.
SetupMemXfer setupmemxfer.hpp	TW_SETUPMEMXFER	
Detail::UserInterface userinterface.hpp	TW_USERINTERFACE	Mimo Windows skrývá položku parent (TW_USERINTERFACE.hParent), která zde nemá význam. Veřejné při implementaci datového zdroje, jinak pouze interní použití.

Tabulka 2: Jednoduché třídy TWAIN++

Název, soubor	Typ TWAIN	Poznámka
AudioNativeXfer audionativexfer.hpp	TW_HANDLE	Obalující třída nad Handle (TW_HANDLE) pro účely přenosu nativních zvukových dat. Handle je uzamčeno po celou dobu životnosti instance. Umožňuje předat handle do správy uživateli. Zajišťuje typovou bezpečnost.
ImageNativeXfer imagenativexfer.hpp	TW_HANDLE	Obdoba třídy AudioNativeXfer pro nativní obrazová data.
CustomData customdata.hpp	TW_CUSTOMDATA	Zjednodušuje uzamčení handle a přístup k paměti pomocí třídy Detail::Lock.
StatusUtf8 statusutf8.hpp	TW_STATUSUTF8	Zjednodušuje zadání obsaženého řetězce, také uzamčení handle a přístup k paměti pomocí třídy Detail::Lock.
JpegCompression jpegcompression.hpp	TW_JPEGCOMPRESSION	Dynamická paměť spravována třídou Memory.

Tabulka 3: Jednoduché třídy TWAIN++ s dynamickou pamětí

Příloha 2

Volby SANE++ v backendu

Volba SANE++	Protějšek v TWAIN++	Poznámka
`tl-x, tl-y, br-x, br-y`	`ImageLayout`	Převod jednotek z palců (TWAIN++) na milimetry (SANE++) a zpět.
`x-resolution`	`CapType::IXResolution`	Náhrada standardní volby `resolution`, která nastavuje rozlišení pouze horizontální osy obrazu.
`y-resolution`	`CapType::IYResolution`	Náhrada standardní volby `resolution`, která nastavuje rozlišení pouze vertikální osy obrazu.
`depth`	`CapType::IBitDepth`	Nestandardní volba nastavující barevnou hloubku obrazu.
`pixel-type`	`CapType::IPixelType`	Nestandardní volba nastavující typ barevné informace. Podporovány pouze typy `PixelType::BlackWhite`, `PixelType::Gray` a `PixelType::Rgb`.

Tabulka 4: Volby SANE++ v backendu

Příloha 3

Volby a operace datového zdroje

Volba TWAIN++	Poznámka
`CapType::IXferMech`	Změna přenosové metody přímo na vzdálené straně je nebezpečná. Síťový protokol podporuje pouze paměťový přenos. Musí být spravováno lokálně.
`CapType::Indicators`	Nastavení ukazatelů přenosu na vzdálené straně nemá smysl pro lokální aplikace. Realizováno lokálně jako nepodporovaná volba.
`CapType::IndicatorsMode`	
`CapType::EnableDsUiOnly`	Zjištění, zda vzdálené GUI lze zobrazit pouze v módu pro nastavení položek nebo jej lze nezobrazit, je taktéž zbytečné. Implementace lokálního GUI vždy podporuje obě možnosti, nastaveno na logickou hodnotu pravda.
`CapType::UiControllable`	
`CapType::SupportedCaps`	Seznam dostupných voleb. Jedná se pouze o odstranění nepodporovaných voleb ze seznamu získaného ze vzdáleného zařízení a zajištění přítomnosti voleb výslovně podporovaných.

Tabulka 5: Lokální volby TWAIN++

Operace TWAIN++	Poznámka
`ArgType::Capability`	Podporovány pouze vyjmenované volby, viz příloha 7.
`ArgType::ExtImageInfo`	Nepodporováno, v SANE neexistuje odpovídající standardní operace nebo volba.
`ArgType::Identity`	Použito k otevření a uzavření spojení a inicializaci voleb.
`ArgType::ImageInfo`	Podpora pouze pro černobílé a barevné (RGB) obrázky a obrázky v odstínech šedi. Data získána z parametrů SANE a rozlišení.
`ArgType::ImageLayout`	Data získána z voleb SANE `tl-x`, `tl-y`, `br-x`, `br-y`, podpora závisí na jejich přítomnosti.
`ArgType::ImageMemXfer`	Postaveno nad datovým spojením SANE. Přenáší vždy nenulový počet celých řádků včetně případné výplně.
`ArgType::Palette8`	Nepodporováno, v SANE neexistuje odpovídající standardní operace nebo volba.
`ArgType::PendingXfers`	Uzavírací operace použity k zavření datového spojení.

Operace TWAIN++	Poznámka
`ArgType::SetupMemXfer`	Data odhadnuta z parametrů SANE.
`ArgType::StatusUtf8`	Nepodporováno.
`ArgType::UserInterface`	Užito k spuštění skenování a otevření datového spojení nebo naopak k jeho uzavření a přerušení skenování.
`ArgType::XferGroup`	Vrací vždy `DataGroup::Image`. SANE nepodporuje přenos zvuku.

Tabulka 6: Operace TWAIN++ nad protokolem SANE

Volba TWAIN++	Poznámka
`CapType::SupportedCaps`	
`CapType::XferCount`	Vyžadovaná volba, nemění nastavení SANE.
`CapType::ICompression`	Vždy vrací `Compression::None`.
`CapType::IBitDepth`	Data získána z parametrů SANE.
`CapType::IBitOrder`	Vždy vrací `BitOrder::MsbFirst`. Ze standardních voleb SANE nelze zjistit pořadí bajtů před přenesením obrazových dat.
`CapType::IPlanarChunky`	Vždy vrací `PlanarChunky::Chunky`. Přenášení obrazu složeného z barevných rámců není podporováno.
`CapType::DeviceOnline`	Zda je spojení se vzdáleným zařízením stále otevřeno.
`CapType::SupportedDats`	
`CapType::IUnits`	Vždy vrací `Unit::Inches`.
`CapType::IPixelFlavor`	Vždy vrací `PixelFlavor::Chocolate`.
`CapType::IPhysicalHeight`	Získáno z výchozích hodnot `tl-x`, `tl-y`, `br-x`, `br-y`, podpora závisí na jejich přítomnosti.
`CapType::IPhysicalWidth`	
`CapType::IXNativeResolution`	Získáno z hodnoty `x-resolution`/`y-resolution`, v případě nepodpory z volby `resolution`. Podpora závisí na přítomnosti aspoň jedné z nich.
`CapType::IXResolution`	
`CapType::IYNativeResolution`	
`CapType::IXResolution`	
`CapType::IPixelType`	Získáno z parametrů SANE, podporován pouze mód RGB a stupně šedi.

Tabulka 7: Volby TWAIN++ nad protokolem SANE

Příloha 4

Operace a volby miniovladače WIA

Příkaz `MicroEntry()`	Poznámka
`CMD_SETSTIDEVICEHKEY`	Nastavení handle do oblasti registru s nastavením.
`CMD_INITIALIZE`	Vytváří instanci struktury `ScanStore` a získává z registru identifikátor odpovídajícího datového zdroje. Je proveden pokus o inicializaci struktury (chyby jsou ignorovány).
`CMD_UNINITIALIZE`	Uvolnění instance `ScanStore`.
`CMD_SETXRESOLUTION`	Nastavuje horizontální/vertikální rozlišení. Inicializuje instanci `ScanStore`, pokud tato operace ještě neproběhla. Podle rozlišení a skenovací oblasti také upravuje hodnoty předávané miniovladači (horizontální pozice oblasti, šířka oblasti, počet pixelů a bajtů na řádek; vertikální pozice oblasti, výška oblasti, počet řádků).
`CMD_SETYRESOLUTION`	
`CMD_SETDATATYPE`	Nastavuje barevný model obrazu a odpovídající bitovou hloubku. Tabulka níže ukazuje právě toto mapování. Nepodporuje-li odpovídající datový zdroj TWAIN barevný model nebo barevná hloubka nesouhlasí, model není podporován na straně mikroovladače.
`CMD_GETCAPABILITIES`	Vrací prázdný seznam.
`CMD_GETSUPPORTEDFILEFORMATS`	Vrací prázdný seznam. Nadřazený miniovladač doplní formát BMP bez souborové hlavičky.
`CMD_GETSUPPORTEDMEMORYFORMATS`	
`CMD_RESETSCANNER`	Žádná operace neprovedena, vrací pouze nechybovou hodnotu `S_OK`, jinak nemůže být mikroovladač použit.
`CMD_SETSCANMODE`	
`CMD_STI_DIAGNOSTIC`	
`CMD_STI_DEVICERESET`	
`CMD_SETINTENSITY`	
`CMD_SETCONTRAST`	

Tabulka 8: Příkazy funkce `MicroEntry()`

Barevný model WIA	Příznak WIA	Barevný model TWAIN++	Barevná hloubka
WIA_DATA_COLOR	SUPPORT_COLOR	`PixelType::Rgb`	24
WIA_DATA_GRAYSCALE	SUPPORT_GRAYSCALE	`PixelType::Gray`	8
WIA_DATA_THRESHOLD	SUPPORT_BW	`PixelType::BlackWhite`	1

Tabulka 9: Mapování barevných modelů

Příloha 5

Operace a volby klienta WIA

Operace TWAIN++	Poznámka
`ArgType::Capability`	Podporována pouze omezená množina voleb. Více v tabulce 11.
`ArgType::ExtImageInfo`	Nepodporováno.
`ArgType::Identity`	Nemá zvláštní význam.
`ArgType::ImageInfo`	Data získána z atributů WIA `WIA_IPS_XRES`, `WIA_IPS_YRES`, `WIA_IPS_XEXTENT`, `WIA_IPS_YEXTENT`, `WIA_IPA_DEPTH`, `WIA_IPA_DATATYPE`, `WIA_IPA_CHANNELS_PER_PIXEL` a `WIA_IPA_BITS_PER_CHANNEL`. Konverzní funkce pro datový typ je uvedena v tabulce 12.
`ArgType::ImageLayout`	Data získána z atributů WIA `WIA_IPS_XRES`, `WIA_IPS_YRES`, `WIA_IPS_XEXTENT`, `WIA_IPS_YEXTENT`, `WIA_IPS_XPOS`, `WIA_IPS_YPOS`.
`ArgType::ImageMemXfer`	
`ArgType::Palette8`	Nepodporováno.
`ArgType::PendingXfers`	Nemá zvláštní význam.
`ArgType::SetupMemXfer`	Metoda `IWiaDataTransfer::idtGetExtendedTransferInfo()` je užita k získání odpovídajících dat.
`ArgType::StatusUtf8`	Nepodporováno.
`ArgType::UserInterface`	Slouží pouze k nastavení příznaků přenosu.
`ArgType::XferGroup`	Vrací vždy `DataGroup::Image`.

Tabulka 10: Operace TWAIN++ nad rozhraním WIA

Volba TWAIN++	Poznámka
`CapType::SupportedCaps`	
`CapType::XferCount`	Vyžadovaná volba, nemění nastavení WIA.
`CapType::ICompression`	Vždy vrací `Compression::None`.
`CapType::IBitDepth`	
`CapType::IBitOrder`	Vždy vrací `BitOrder::MsbFirst`.
`CapType::IPlanarChunky`	Vždy vrací `PlanarChunky::Chunky`.
`CapType::DeviceOnline`	Atribut `WIA_DPA_CONNECT_STATUS`.
`CapType::SupportedDats`	
`CapType::IUnits`	Vždy vrací `Unit::Inches`.
`CapType::IPixelFlavor`	Vždy vrací `PixelFlavor::Chocolate`.
`CapType::IXNativeResolution`	Atribut `WIA_DPS_OPTICAL_XRES`.
`CapType::IYNativeResolution`	Atribut `WIA_DPS_OPTICAL_YRES`.
`CapType::IPhysicalHeight`	Atribut `WIA_DPS_VERTICAL_BED_SIZE`.
`CapType::IPhysicalWidth`	Atribut `WIA_DPS_HORIZONTAL_BED_SIZE`.
`CapType::IXResolution`	Atribut `WIA_IPS_XRES`.
`CapType::IYResolution`	Atribut `WIA_IPS_YRES`.
`CapType::IPixelType`	Atribut `WIA_IPA_DATATYPE`. Konverzní funkce je uvedena v tabulce 12.

Tabulka 11: Podporované volby TWAIN++ nad rozhraním WIA

Barevný typ TWAIN++	Barevný typ WIA
`PixelType::Rgb`	`WIA_DATA_COLOR` Následující typy se uplatní pouze ve směru WIA→TWAIN++: `WIA_DATA_COLOR_DITHER`, `WIA_DATA_COLOR_THRESHOLD`, `WIA_DATA_DITHER`.
`PixelType::Gray`	`WIA_DATA_GRAYSCALE`
`PixelType::BlackWhite`	`WIA_DATA_THRESHOLD`

Tabulka 12: Převodní funkce barevných typů

Příloha 6

DVD se zdrojovými soubory

Obsah přiloženého DVD

- `/src` – adresář se zdrojovými soubory programové části,
- `/bin` – adresář s instalátory klienta a serveru pro Windows a archivem s backendem pro Linux,
- `/dip.pdf` – technická zpráva ve formátu PDF,
- `/dip.docx` – technická zpráva ve zdrojovém formátu DOCX,
- `/install.txt` – návod k instalaci na Windows a Linuxu,
- `/build.txt` – návod k přeložení programové části.