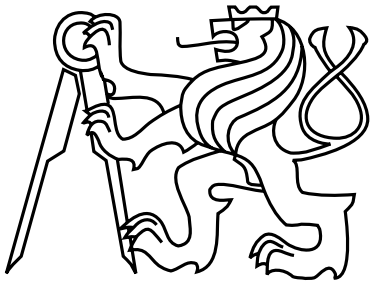
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Komunikační platforma pro WEB

Michal Macháček

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

25. května 2012

Poděkování

Děkuji všem, kteří mě trpělivě podporovali po dobu mého studia. Děkuji vedoucímu své bakalářské práce za jeho osobní přístup i užitečné odborné rady.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Příbrami dne 24. 5. 2012 .....

Abstract

This thesis describes an instant messaging library libpurple and a voice and video calling library libjingle. Basic functions of these libraries are made accessible from the web browser via NPAPI plugin. The plugin is created by the FireBreath toolkit. The result of this work is a prototype of a web user interface usable for basic text, audio and video communication. All implementation details are well documented so that the work can be easily used in the further development.

Abstrakt

Tato práce popisuje instant messaging knihovnu libpurple a knihovnu libjingle pro hlasové a video volání. Základní funkce těchto knihoven jsou zpřístupněny webovému uživatelskému rozhraní přes NPAPI plugin. Plugin je vytvořen pomocí toolkitu FireBreath. Výstupem této práce je demonstrační webové uživatelské rozhraní použitelné pro základní textovou, audio a video komunikaci. Podrobnosti implementace jsou důkladně zdokumentovány, aby bylo možné použít práci v dalším vývoji.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Jednou ze základních funkcí Internetu je komunikace mezi uživateli. Možností jak komunikovat je celá řada. V této práci bude kladen důraz na textovou komunikaci pomocí krátkých zpráv *(instant messaging)*, hlasovou a video komunikaci. Existujících řešení je mnoho, bohužel však většinou nejsou mezi sebou kompatibilní. To představuje pro uživatele nepohodlí, protože musí mít spuštěno více klientů a přecházet mezi nimi podle toho, jaký protokol používá druhá strana. Nedá se přitom očekávat, že sjednocující iniciativa přijde od výrobců daných technologií, protože nebudou chtít byť nepřímo napomáhat konkurenčnímu produktu. Výjimku by mohly tvořit snad jen akvizice nebo smlouvy o spolupráci. Například akvizice Skype společností Microsoft a spolupráce Microsoftu s Facebookem a společností Nokia by teoreticky mohly dát vzniknout jednotné komunikační koncepci napříč jejich produkty a weby. Zatím tomu tak ovšem není. Jistých úspěchů bylo dosud dosaženo v oblasti textové komunikace zavedením XMPP protokolu a aplikací, které podporují více protokolů pro instant messaging a umožňují je používat zároveň.

Tyto problémy se snaží řešit projekt ExtBrain [38]. V jeho rámci vzniká projekt Extbrain Communicator, který bude umožňovat z jednoho místa komunikovat pomocí mailu, krátkých textových zpráv, hlasových a video hovorů, diagramů a dalších. Cílem této práce je prozkoumat možnosti vybraných komunikačních knihoven, které by byly použitelné v ExtBrain Communicatoru pro přenos textových zpráv a hlasové i video hovory. Nad těmito knihovnami dále navrhnout a napsat základní webové rozhraní, které bude umožňovat jejich používání. Rozhraní bude vytvořeno pomocí HTML a JavaScriptu. Ke knihovnám se z něj bude přistupovat přes NPAPI plugin [11] vytvořením instance pluginu ve stránce a voláním jeho API. Toto rozhraní bude vstupem pro další práci jiného řešitele, který jej vizuálně přizpůsobí stylu použitému v Extbrain Communicatoru a zaintegruje jej do něho. Celá práce proto musí být zdokumentována tak, aby tento proces řešiteli maximálně usnadnila.

V kapitole 2 bude upřesněn popis problému a navrhovaného řešení. Bude provedena stručná rešerše existujících řešení a jejich srovnání s navrhovaným. Kapitola 3 se zaměří na popis knihoven a technologií tvořících infrastrukturu řešení a na samotné knihovny libpurple a libjingle. Budou popsány jejich obecné vlastnosti a funkce tak, aby byl položen teoretický základ pro implementaci. Samotná implementace bude navržena a popsána v kapitole 4, zvlášť pro libpurple a libjingle plugin. Testování implementovaných aplikací bude popsáno v kapitole 5. Závěrečné zhodnocení kladů a nedostatků práce bude provedeno v kapitole 6, kde bude rovněž diskutováno možné pokračování práce.

Kapitola 2

Popis problému a specifikace cíle

V této kapitole bude popsán aktuální stav projektu a rešerše existujících řešení. Bude upřesněn popis cíle a navržen postup řešení. Bude provedeno srovnání existujících řešení s navrhovaným.

2.1 Aktuální stav

Extbrain Communicator již částečnou podporu pro komunikaci obsahuje. Byl vytvořená JavaScriptová implementace protokolu XMPP a integrována knihovna PJSip pro SIP volání. Chybí ovšem ostatní IM protokoly, zejména u nás poměrně rozšířené ICQ. Podpora videohovorů zatím chybí úplně.

2.2 Specifikace cíle

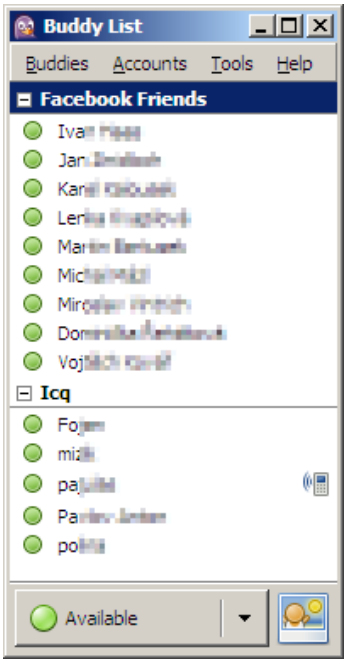
Po dohodě s vedoucím projektu bylo rozhodnuto, že tato práce se bude pro textovou komunikaci věnovat knihovně libpurple [42] [52] a pro volání knihovně libjingle. Cílem tedy bude vytvoření doplňku, který bude provádět instant messaging přes knihovnu liburple a volání přes knihovnu libjingle. Pro IM bude kladen důraz na protokol ICQ, který zatím implementován neni. Přenos zvuku a videa bude vyvíjen testován na komunikaci s Google chatem (libjingle je vyvíjena společností Google).

2.3 Existující řešení

Návrhu architektury a implementace musí předcházet rešerše existujících řešení použitelných pro internetovou komunikaci. Jejím cílem bude kromě zmapování současného stavu také analýza výhod a nevýhod jednotlivých produktů. V současné době je k dispozici několik základních kategorií řešení.

2.3.1 Aplikace pro instant messaging

Do této kategorie spadají klientské programy, které si uživatel spouští ve svém operačním systému. Obstarávají veškerou komunikaci se serverem dané služby, lokálně uchovávají přihlašovací údaje a hesla. Existují programy jednostranně zaměřené na jediný protokol, většinou poskytované provozovatelem dané služby. Typickým příkladem je oficiální ICQ klient. Oproti tomu existují i programy zvládající celou řadu protokolů, pocházející většinou od nezávislých vývojářů. Kromě toho, že umožňují zároveň používat více protokolů, nabízejí většinou možnost seskupovat kontakty z těchto jednotlivých protokolů podle fyzických osob ke kterým patří. Do této skupiny patří například klienti Instantbird, Pidgin (obr. 2.1) a další. Oba jsou postaveny nad knihovnou libpurple a jsou distribuovány jako open-source, což bude zdrojem neocenitelných poznatků při dalším vývoji. Inspiraci bude možné čerpat i ze způsobu implementace pro různé platformy.



Obrázek 2.1: IM program Pidgin

2.3.2 Chaty přidružené webům

Další kategorií jsou chaty přidružené webům nebo sociálním sítím. Sociální sítě zaznamenaly v poslední době prudký vzestup a uživatelé chtějí mít možnost kromě zveřejňování svých statusů také komunikovat s konkrétní osobou soukromě. Vzhledem k velkému množství času, který na těchto sítích mnoho lidí tráví, je nutno s jejich komunikačními protokoly počítat jako s rovnocennou alternativou ke starším IM protokolům. Typickými zástupci této kategorie jsou Google chat (obr. 2.1) nebo Facebook chat (obr. 2.3). Oba shodně podporují textovou konverzaci i audio a video hovory. Autentizace uživatele probíhá přes web nadřazené služby.

Textovou komunikaci zvládají přímo v prohlížeči, pro audio a video hovory je potřeba mít nainstalovaný příslušný plugin.

2.3.3 Aplikace pro videohovory

Dominantním představitelem této kategorie je Skype. Je to velmi rozšířený software umožňující kromě IM a videohovorů také hovory do klasické telefonní sítě. Jeho velkou nevýhodou je uzavřený protokol, který znemožňuje vývoj alternativních klientů nebo integraci do existujících programů. Další nevýhodou je fakt, že v případě veřejné IP adresy bude program sloužit jako server pro potřeby ostatních uživatelů služby.

2.3.4 Web-based messengery

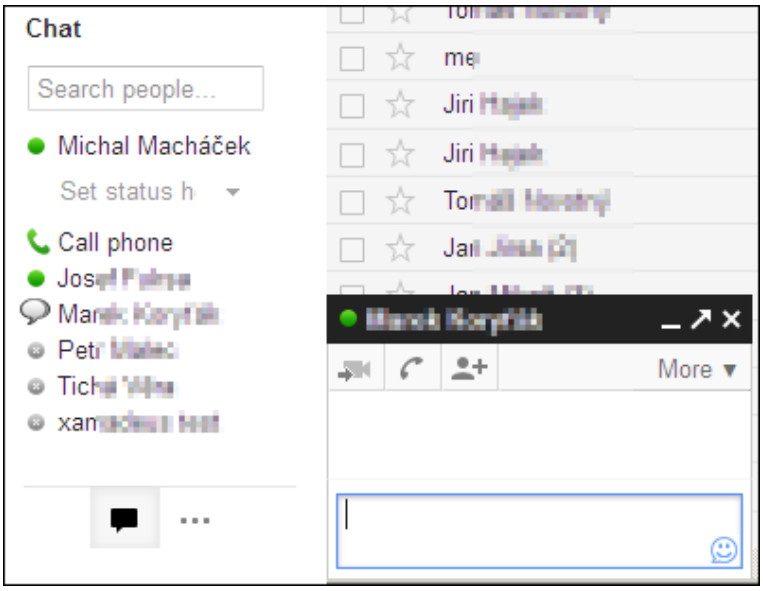
Sem spadají webové systémy, kterým uživatel svěří přístupové údaje ke svým službám a přistupuje k nim pomocí webového rozhraní. Výhodou tohoto řešení je to, že uživatel nemusí instalovat žádný software a má svoje účty k dispozici na libovolném počítači. Tento přístup navíc šetří výpočetní výkon a objem přenášených dat, což může být důležité zvláště u mobilních zařízení. Potencionálním rizikem může být skutečnost, že svoje přístupové údaje k jednotlivým účtům svěřujeme třetí straně. Patří sem systémy jako Meebo, immo.im, eBuddy (obr. 2.4).

2.4 Srovnání s navrhovaným řešením

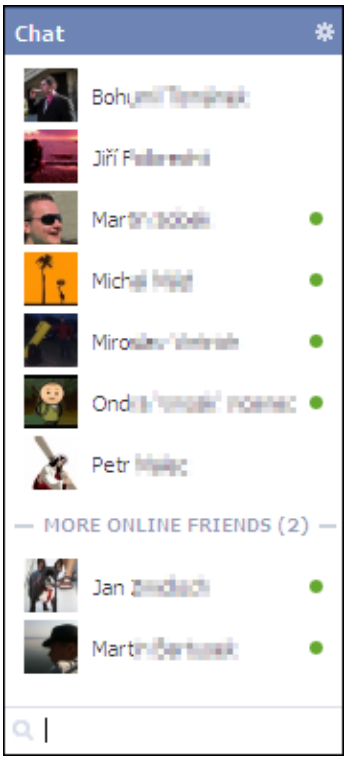
Výstupy této práce mají fungovat jako pluginy v internetovém prohlížeči. To znamená, že nebude potřeba instalovat žádnou další aplikaci, nicméně režie spojená s během klienta se tím pouze přesune do příslušného pluginu. Přihlašovací údaje se budou ukládat pouze na lokálním stroji. Použití osvědčené knihovny libpurple zpřístupní stejnou škálu protokolů, jakou disponují zmíněné IM programy. Uživatelské rozhraní psané v HTML a JavaScriptu bude ze své podstaty silně platformně nezávislé. Použité knihovny a technologie jsou navrženy jako multiplatformní, což by mělo zásadně snížit náročnost implementování pod různými operačními systémy.

2.5 Struktura práce

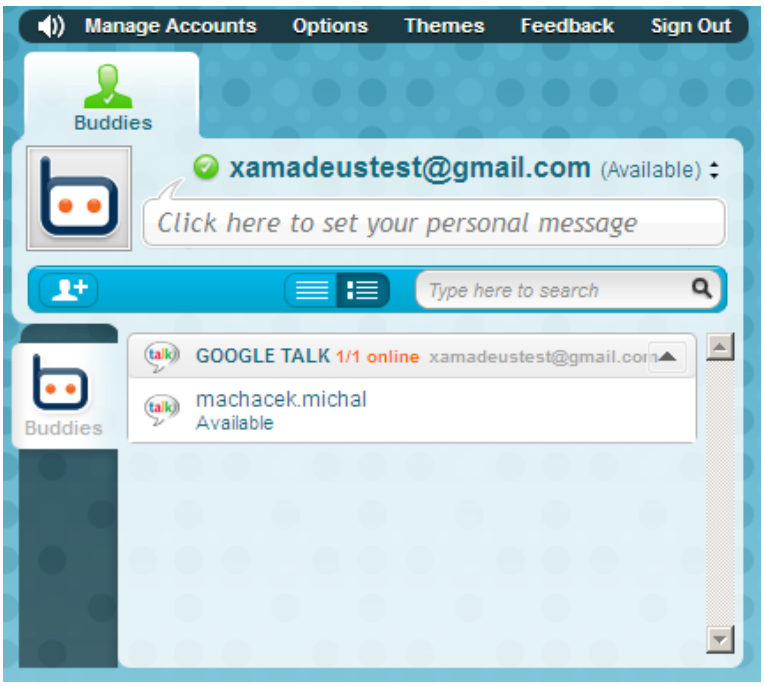
Práce začne podrobným popisem použitých knihoven a technologií. Důraz bude kladen zejména na ty části, které budou reálně využity při implementaci. Na základě této analýzy bude navržena základní architektura řešení. Na jejím základě poté budou implementovány a zdokumentovány požadované doplňky. Při popisu budou zvlášť zdůrazněny možné problematické nebo nestandardní části řešení, které se objevily během implementace a způsobovaly různé problémy. To ušetří velké penzum práce budoucím řešitelům navazujících projektů. Na závěr bude provedeno zhodnocení práce a posouzení funkčnosti. Budou také nastíněny možné směry dalšího vývoje.



Obrázek 2.2: Chat přidružený ke službám Google



Obrázek 2.3: Chat přidružený k sociální sítí Facebook



Obrázek 2.4: Webové rozhraní eBuddy

Kapitola 3

Popis knihoven a technologií

Tato kapitola se bude zabývat popisem technologií a knihoven, které budou tvořit infrastrukturu řešení. Nejprve budou popsány základní typy doplňků do prohlížečů a jejich vlastnosti. Následovat bude popis toolkitu FireBreath, který usnadňuje tvorbu těchto doplňků. Klíčovou částí bude popis samotných knihoven libpurple a libjingle.

3.1 Možnosti rozšíření funkcí prohlížeče

Do současných internetových prohlížečů je možno dodatečně doinstalovat řadu doplňků, které rozšiřují jeho funkcionalitu, usnadňují práci nebo jen mění vizuální vzhled. Pro vysvětlení základních pojmů bude použita terminologie z prohlížeče Mozilla Firefox, rozdíly a specifika implementace nebo instalace pro ostatní prohlížeče budou rozvedeny v implementační části práce. Základní principy však jsou obecně platné.

Souhrnný název pro rozšíření všech zmíněných kategorií je *add-ons*, v češtině se používá výraz *doplňky* [31]. Doplňky se dále dělí do následujících kategorií.

- Extensions (rozšíření)
- Themes (vizuální motivy)
- Plugins (pluginy, zásuvné moduly)

Přehled aktuálně nainstalovaných pluginů lze získat buď přes příslušné menu nebo pomocí `about:plugins` či `about:addons`. V této práci budou důležitá rozšíření a pluginy.

3.2 Rozšíření

Rozšíření dodávají prohlížeči novou funkcionalitu, přičemž využívají pouze jeho vlastní technologie na tvorbu GUI a skriptování. V tomto případě jde o XUL, který umožňuje měnit nebo doplňovat GUI, a JavaScript, ve kterém se píší obslužné skripty. Podrobný popis je na [18]. Díky použití těchto technologií jsou rozšíření silně platformně nezávislá (s výjimkou v případě že obsahují binární komponenty). Samotná instalace je velmi jednoduchá i pro

laické uživatele, spočívá ve stažení jednoho souboru .xpi a potvrzení. Právě z tohoto důvodu bude v této práci rozšíření použito jako prostředek ke snadné instalaci pluginu. Ostatní jeho možnosti využity nebudou.

3.3 Plugin

Pluginy jsou sdílené knihovny, které umožňují zobrazit obsah, na který nestačí vlastní možnosti prohlížeče nebo rozšíření [21]. To je hlavní rozdíl oproti rozšíření a vyplývají z něho důležité vlastnosti pluginů.

3.3.1 Závislost na platformě

Aby bylo možné hovořit o meziplatformní podpoře, musí být knihovny pluginu buildovány a distribuovány zvlášť pro každou platformu, na které prohlížeč běží. V praxi mohou být použita dvě možná řešení tohoto problému :

- Vytvořit instalátor pro každou platformu zvlášť a nechat uživatele stáhnout ten správný pro jeho systém. Stránka přitom může na základě detekce typu operačního systému doporučit konkrétní variantu. Drobnou nevýhodou je mírné nepohodlí, které by však nemělo činit problém ani laickým uživatelům. Výhodou je menší velikost balíčku, který nemusí obsahovat nepoužitelné binární knihovny pro jiné systémy.

- Zabalit všechny verze knihoven do jednoho balíčku s tím, že prohlížeč bude ignorovat verze pro cizí operační systémy. Výhodou je jeden univerzální instalátor pro všechny platformy, nevýhodou větší objem díky knihovnám pro ostatní systémy.

V této práci se bude používat druhý ze zmíněných způsobů. Konkrétní formáty instalačních balíčků a specifika pro různé operační systémy budou popsány v implementační části práce.

3.3.2 Vlastní proces

Specifickou a důležitou vlastností pluginů je to, že běží ve zvláštním procesu odděleném od prohlížeče. Na jednu stranu je to výhodné v tom, že pád pluginu neovlivní ostatní otevřená okna. Je také možné se na proces připojit debuggerem, což velmi usnadňuje ladění. Rizikem je však to, že externí binární kód se nedá uhlídat bezpečnostními mechanismy prohlížeče. Má přitom plná práva aktuálního uživatele. Na tuto skutečnost důrazně upozorňuje např. [11], kde je doporučeno použití pluginu až jako nejkrajnější možnosti když vše ostatní selže.

V této práci bude přesto NPAPI pluginu použito. V předchozím semestrálním projektu autora této práce i bakalářské práce Yuna Ruana [36] se ukázalo, že cesta použitím rozšíření volající binární komponenty přes XPCOM [23] není pro použití v současných verzích prohlížečů vhodná. Navíc projekt ExtBrain Communicator celkově opouští cestu XUL uživatelského rozhraní a přechází na HTML, zejména z důvodu přenositelnosti.

3.4 NPAPI plugin

NPAPI (Netscape Plugin Application Programming Interface) je multiplatformní architektura pluginů používaná ve většině hlavních prohlížečů. Původně byla vyvinuta pro Netscape Navigator, nyní je podporována v prohlížečích Mozilla Firefox, Google Chrome, Microsoft Internet Explorer a dalších [11],[35]. Základním zdrojem informací je Gecko Plugin API Reference na Mozilla Developer Network [20].Na implementaci pluginu právě pro tyto prohlížeče bude zaměřena tato práce.

3.4.1 Obecný popis

Každý plugin je zaregistrován pod jedním nebo více MIME typy [30]. V seznamu nainstalovaných pluginů (`about:plugins`) lze zjistit jejich podrobný seznam. Pokud uživatel v prohlížeči otevře stránku vyžadující plugin, provedou se následující kroky :

- Vyhledá se plugin s odpovídajícím MIME typem
- Kód pluginu se nahraje do paměti
- Provede se inicializace pluginu
- Vytvoří se instance

Je možné mít na jedné stránce více instancí pluginu stejného typu i naopak více instancí různého typu. Kód pluginu je udržován v paměti tak dlouho, dokud je aktivní alespoň jedna instance. Poté se z paměti uvolní. Pokud není plugin aktuálně nahrán, nekonzumuje žádné systémové zdroje (kromě místa na disku).

3.4.2 Registrace pluginů

Při startu prohlížeče se podle platformy prohledávají v přesně daném pořadí různá místa a registrují se pluginy v nich umístěné. Seznámení se s nimi je důležité pro pochopení různých možností instalace pluginu, které budou diskutovány v implementační části práce. Zjistíme, že každá z těchto možností se váže k některému ze standardně prohledávaných míst.

Windows

- Adresář z proměnné prostředí `MOZ_PLUGIN_PATH`
- `%APPDATA%\Mozilla\plugins`, kde `%APPDATA%` je proměnná s cestou do uživatelského aplikačního adresáře
- `Application directory\plugins`, kde `Application directory` je instalační adresář aplikace
- Pluginy které jsou součástí rozšíření
- `Profile directory\plugins`, kde `Profile directory` je adresář uživatelského profilu

- Adresáře uvedené v `HKEY_CURRENT_USER\Software\MozillaPlugins\*\Path`, kde `*` představuje jméno pluginu
- Adresáře uvedené v `HKEY_LOCAL_MACHINE\Software\MozillaPlugins\*\Path`, kde `*` představuje jméno pluginu

Mac OS X

- `Application directory/plugins`, kde `Application directory` je instalační adresář aplikace
- `~/Library/Internet Plug-Ins`
- `/Library/Internet Plug-Ins`
- `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Resources`
- Pluginy které jsou součástí rozšíření
- `Profile directory/plugins`, kde `Profile directory` je adresář uživatelského profilu

Linux

- Adresář z proměnné prostředí `MOZ_PLUGIN_PATH`
- `~/.mozilla/plugins`
- `Application directory/plugins`, kde `Application directory` je instalační adresář aplikace
- `/usr/lib/mozilla/plugins` or `/usr/lib64/mozilla/plugins` on 64bit systems
- Pluginy které jsou součástí rozšíření
- `Profile directory/plugins`, kde `Profile directory` je adresář uživatelského profilu

3.4.3 Plugin side API

V kapitole Plug-in side API [22] na Mozilla Developer Network je specifikována sada funkcí, které jsou pluginem implementovány a mohou být volány prohlížečem. V rámci této práce se jimi nebudeme zvlášť zabývat, tato vrstva je plně v režii použitých technologií. Pro JavaScriptová volání knihovních funkcí pluginu (tedy funkcí binární knihovny, která je pomocí pluginu zpřístupňována, nikoliv funkcí z plugin side API) bude použito rozhraní zpřístupněné toolkitem FireBreath.

3.5 Firebreath toolkit

3.5.1 Obecný popis

FireBreath [27] je framework usnadňující tvorbu pluginů do prohlížečů. Jím vytvořené pluginy jsou typy NPAPI. Použitým programovacím jazykem pro psaní pluginů je C++. FireBreath klade velký důraz na multiplatformní podporu. Jednak tím, že jeho infrastruktura je tvořena sadou shellových a python skriptů, které jsou snadno přenositelné do různých systému. Dále pak tím, že pro zdrojové kódy doplňku (zvlášť uložené a společné pro všechny platformy) dokáže pro každou z cílových platforem automaticky vygenerovat CMake nebo Visual Studio projekt. Kromě společného nastavení parametrů pro generování se dají zvlášť nastavit specifické vlastnosti pro jednotlivé cílové operační systémy.

3.5.2 Instalace toolkitu

Instalace je velmi jednoduchá, spočívá pouze v rozbalení archivu nebo stažení aktuální verze z repozitáře. Vše je dobře popsáno na stránkách projektu [27]. Pozornost je třeba věnovat tomu, aby byl nainstalován Python v některé z požadovaných verzí. Pro buildování pluginu ve Visual Studiu je samozřejmě třeba mít nainstalovanou některou z jeho podporovaných verzí. Totéž platí v případě CMake projektu.

3.5.3 Vytvoření nového pluginu

Nový projekt se vytváří spuštěním připraveného skriptu pomocí příkazu

```
python fbgen.py
```

Jako první bude uživatel tázán na několik informací ohledně jména, popisu a společnosti pluginu. Tyto informace se budou zobrazovat v prohlížeči v seznamu nainstalovaných pluginů. Určuje se zde i MIME typ, který je klíčový pro identifikaci pluginu (viz 3.4.1). Po zadání všech údajů se vygeneruje do adresáře `projects` základní sada zdrojových kódů doplňku a konfiguračních souborů pro generování build projektu.

- `PluginConfig.cmake` obsahuje většinu informací získaných v předchozím kroku, lze je dodatečně měnit
- `CMakeLists.txt` je konfigurace pro generování build projektu společná pro všechny platformy
- `Factory.cpp` implementuje třídu, která vytváří, inicializuje a ruší objekty instancí pluginu
- `NazevNovehoPluginu.cpp` kód pluginu který se bude instancovat
- `NazevNovehoPluginuAPI.cpp` je třída, která bude implementovat metody a funkce, které budou viditelné a volatelné z JavaScriptu.
- `projectDef.cmake` je konfigurace pro generování build projektu specifická pouze pro Windows. Analogické soubory jsou také v adresářích `Mac` pro Mac OS a `X11` pro Linux.

3.5.4 Přidávání API funkcí

Do API lze přidávat vlastní funkce, vlastnosti nebo eventy. Výchozím souborem je třída `NazevNovehoPluginuAPI`. Způsob použití je následující.

- Zpřístupnění členské metody `&PurpleBreathAPI::Send` pro volání z JavaScriptu jako `Send` se provádí v konstruktoru makrem

  ```
  registerMethod("Send", make_method(this, &PurpleBreathAPI::Send));
  ```

- Zpřístupnění Property (vlastnosti), což je položka kterou lze číst, měnit nebo obojí.

  ```
  registerProperty("status", make_property(this, &MyPluginAPI::get_fullName,
  &MyPluginAPI::set_fullName));
  ```

  Podrobnější specifikace na [1].

- Vytváření eventu se provádí v public části deklarace třídy makrem

  `FB_JSAPI_EVENT(connection\error,1, (std::string));`,

  kde číslovka označuje počet argumentů a následuje jejich čárkou oddělený seznam.

Jako návratové typy a typy argumentů lze použít širokou škálu podporovaných JSAPI typů [2]. Kromě obvyklých primitivních typů je důležitý typ `FB::Variant`, který může reprezentovat libovolný datatyp tak, jak je to obvyklé v JavaScriptu. Pokud je třeba předávat v proměnné více prvků, lze použít `FB::VariantList` nebo `FB::VariantMap`, což je pole, respektive asociované pole prvků typu `FB::Variant`.

3.5.5 Build nového pluginu

Aby bylo možno plugin zbuildovat, je ještě potřeba vygenerovat z jeho zdrojových kódů CMake nebo Visual Studio projekt. To se provádí pod Windows pomocí skriptu

`prep2010.cmd`

respektive pomocí skriptu s označením odpovídajícím požadované verzi Visual Studia. Pro vygenerování linuxového CMake projektu slouží skript

`prepmake.sh`

K dipozici jsou případně i skripty generující projekt pro Eclipse nebo Mac. Nyní je v adresáři `build\projects\NazevNovehoPluginu` připraven solution soubor `.sln` a projekty do Visual Studia nebo CMake projekt. Je potřeba uvědomit si důležité aspekty tohoto mechanismu.

- V tomto adresáři se nacházejí pouze soubory tvořící infrastrukturu projektu, jako například `.sln`, `.vcxproj`, `Makefile` a další. Samotné zdrojové kódy, patřící do těchto projektů, jsou umístěny v podadresáři `projects` hlavního adresáře toolkitu.

- Veškeré nastavení projektů, jako například include adresáře, kompilační flagy, cesty k externím linkovaným knihovnám by mělo být specifikováno v konfiguračních souborech pro generování projektu. To zaručí, že po smazání vygenerovaného projektu nebo přenesení zdrojových kódů pluginu na jiný počítač bude možné získat opět plně funkční projekt.

- Změny v nastavení projektu, provedené ve vygenerovaném projektu, se při přegenerování projektu ztratí. Změny ve zdrojových kódech se promítají přímo do výchozích souborů, ty tedy zůstanou.
- V případě dodržení těchto zásad stačí zálohovat nebo distribuovat obsah výchozího adresáře pro generování, tedy `projects\NazevNovehoPluginu` v hlavním adresáři toolkitu.

Zkompilovaný plugin ve formě dynamické knihovny se nachází v adresáři `build\bin`. Název souboru začíná prefixem `np`, přípona je standardní pro dynamické knihovny v daném systému. Pro Windows `.dll`, pro Linux `.so`. Tento soubor je hlavním výstupem celého vývojového procesu v toolkitu FireBreath. Připraven je i jednoduchý HTML soubor se zabudovaným objektem pluginu a příkladem jednoduchého volání. Před popisem použití pluginu v prohlížeči ještě přehledně zachytíme výslednou strukturu adresářů.

3.5.6 Adresářová struktura toolkitu

Výsledná adresářová struktura FireBreath toolkitu po vytvoření a vygenerování projektu je znázorněna v následujícím výpisu. Uvedeny jsou pouze vybrané adresáře a soubory související s tvorbou a generováním projektů. Pro jednoduchost jsou vypuštěny některé vnořené adresáře se zřejmým účelem. Příklad ilustruje vygenerovaný Visual Studio projekt. V případě CMake projektu zůstává adresářová struktura stejná, liší se pouze souborech v adresáři `build\projects`.

```
Firebreath root dir
├── build
│   ├── bin
│   │   └── npNazevNovehoPluginu.dll
│   └── projects
│       └── NazevNovehoPluginu
│           ├── gen
│           │   └── FBControl.htm
│           └── NazevNovehoPluginu.sln
├── projects
│   └── NazevNovehoPluginu
│       ├── Win
│       │   ├── WiX
│       │   └── projectDef.cmake
│       ├── X11
│       │   └── projectDef.cmake
│       ├── PluginConfig.cmake
│       ├── NazevNovehoPluginu.cpp
│       ├── NazevNovehoPluginuAPI.cpp
│       └── CMakeLists.txt
├── prep2010.cmd
└── fbgen.py
```

3.5.7 Instalace pluginu do prohlížeče

3.5.7.1 Instalace pomocí regsvr32

Hotový plugin je potřeba nainstalovat do prohlížeče, aby jej mohla HTML stránka volat pomocí JavaScriptu. Před samotnou instalací je potřeba zajistit, aby knihovna pluginu měla dostupné všechny knihovny, na kterých je závislá. Existují nástroje, které umí rekurzivně zobrazit závislosti knihoven a jejich umístění v systému, případně hlášení o neúspěchu. Pro Windows je to program Dependency Walker (<`www.dependencywalker.com`>), pod Linuxem lze použít standardní utilitu `ldd` [14]. Důležitá jsou rovněž umístění, ve kterých se knihovny hledají, a jejich pořadí. Jedná se o standardní mechanismy obecně platné pro dynamicky linkované knihovny, dokumentace například pro Windows [16] a pro Linux [15]. Konkrétní specifika týkající se závislostí knihoven pluginu budou upřesněna v implementační části práce.

Jak bylo uvedeno v 3.4.2, jednotlivé možnosti instalace korespondují s umístěními standardně prohledávanými prohlížečem při registraci pluginů.

3.5.7.2 Instalace pomocí regsvr32

Instalace pomocí nástroje `regsvr32` [17] je možná pouze v prostředí Windows. `Regsvr32` umožňuje registrovat a rušit registraci pluginu formou zápisu do registru (viz 3.4.2). Instalace se provádí jednoduše z adresáře obsahujícího knihovnu doplňku pomocí příkazu

```
regsvr32 npKnihovnaDoplnku.dll
```

Odinstalace se provádí stejným způsobem s přepínačem `/u`

```
regsvr32 /u npKnihovnaDoplnku.dll
```

Po úspěšném provedení registrace (žádné hlášení o úspěchu se nezobrazuje) je plugin dostupný pro všechny podporované prohlížeče. Samotná knihovna se nikam nekopíruje, používá se ta z adresáře ve kterém proběhla registrace.

Tento způsob není pro reálné nasazení vhodný, neboť by nutil uživatele při instalaci používat příkazovou řádku. Navíc selhává v případě portable distribuce prohlížeče například na USB úložišti. Naopak velmi výhodný je pro vývojové účely pokud je plugin zaregistrován přímo z výstupního build adresáře. V takovém případě je hned po buildu v prohlížeči bez dalších zásahů automaticky nová verze. Drobnou komplikací může být fakt, že je před buildem potřeba zavřít všechna okna prohlížeče, pokud v nich je plugin aktivní, aby nebyl soubor zamčený pro zápis.

3.5.7.3 Instalace pomocí WiX

Další možností určenou pouze pro prostředí Windows je WiX instalátor. WiX (Windows Installer XML toolset) je sada nástrojů umožňující tvorbu MSI instalátorů podle nastavení specifikovaném ve formátu XML [13]. Projekt instalátoru je v rámci FireBreath toolkitu vytvořen při generování projektu ze zdrojových kódů a je zahrnut do vytvářeného solutionu. Podmínkou je v systému nainstalovaný WiX a konfigurace v souboru `PluginInstaller.wxs`, který je v podadresáři `Win\Wix` ve zdrojových kódech projektu, přehledně viz 3.5.6.

Instalace probíhá formou zápisu do registru (viz 3.4.2) a samotná knihovna se rozbaluje do místa specifikovaného instalátorem. Opět je plugin použitelný ze všech podporovaných prohlížečů. V této práci přichází metoda WiX instalátoru pouze pro prohlížeč Internet Ex-

plorer, protože do Firefoxu a Chrome lze použít ještě jednodušší způsob instalace, navíc multiplatformní,a to pomocí instalátoru rozšíření (XPI) se zabudovaným pluginem.

3.5.7.4 Instalace pomocí XPI

XPI (Cross-Platform Installer Module) [24] je zip archiv obsahující instalátor založený na technologii XPInstall [25]. Tato technologie se používá v aplikacích nad platformou Mozilla k instalování rozšíření. Instalátor je multiplatformní. Samotná instalace je velmi jednoduchá, spočívá ve stažení nebo otevření `.xpi` souboru a potvrzení. XPI může být použit i pro instalaci pluginu, čehož bude v této práci využito. Cílový adresář pro instalaci je automaticky nastaven do aktuálního profilu prohlížeče daného uživatele. To umožňuje selektivně instalovat pouze do konkrétních instalací prohlížeče a řeší problém s jeho portable verzí.

XPI primárně slouží k instalaci rozšíření, která doplňují nebo mění vzhled a vlastnosti prohlížeče. K těmto účelům jej v této práci nebude třeba, popsána bude pouze část týkající se instalace pluginu. Tím se popis i výsledná struktura XPI výrazně zjednoduší.

XPI můžeme buď vytvořit ručně, nebo si nechat jeho kostru vygenerovat online nástrojem [19]. Důležitý je zejména soubor `install.rdf`, ve kterém se definují název a popisy rozšíření. Důležitá je specifikace aplikace a její verze, pro kterou je instalátor určen. Z následujícího fragmentu vyplývá, že instalátor je určen pro Mozilla Firefox verzí 7.0 až 13.0a1. Při pokusu instalovat do jiné verze se objeví chybové hlášení o nekompatibilitě s danou verzí a instalace nebude povolena. Další důležité nastavení je atribut `unpack`, který nastavuje, zda se bude archiv rozšíření v cílovém adresáři rozbalovat. Protože v této práci bude použito pluginů, které budou záviset na dalších knihovnách, musí být tento atribut nastaven na `true`.

```
<em:targetApplication>
    <Description>
      <!-- Firefox -->
      <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
      <em:minVersion>7.0</em:minVersion>
      <em:maxVersion>13.0a1</em:maxVersion>
    </Description>
  </em:targetApplication>
<em:unpack>true</em:unpack>
```

Samotnou knihovnu pluginu, spolu s případnými závislostmi nebo dalšími soubory (například konfiguračními) umístíme do adresáře `plugins`. Prohlížeč ji v takovém případě při startu vyhledá a zaregistruje (viz 3.4.2). Výsledná vnitřní struktura XPI archivu bude vypadat takto (vypsány pouze adresáře týkající se instalace pluginu).

```
PluginInstaller.xpi
├── plugins
│   └── npPluginName.dll
├── chrome.manifest
└── install.rdf
```

3.5.8 Interakce s HTML a JavaScriptem

Ukázkový příklad toho, jak je objekt pluginu zabudován do HTML stránky a jak se provádí volání jeho funkce, je vygenerován do souboru FBControl.htm (viz 3.4.2). Z něho lze vycházet při tvorbě složitějšího rozhraní. Objekt pluginu je do stánky zabudován následujícím způsobem.

```
<object id="plugin0" type="application/x-purplebreath"
        width="300" height="300">
    <param name="onload" value="pluginLoaded" />
</object>
```

Důležitá je specifikace MIME typu pluginu, podle kterého se bude objekt instancovat, v tomto případě `application/x-purplebreath`. Atribut `id` rozlišuje případné další instance. Dále je nastavena velikost zobrazení ve stránce (300×300). Na událost `onload`, tedy dokončení tvorby instance pluginu, je navázána funkce `pluginLoaded`, která umožňuje definovat příslušnou reakci, například hlášení o úspěšném nahrání pluginu. Nyní můžeme k pluginu přistupovat z JavaScriptu jako k běžnému elementu. Přístup lze usnadnit pomocí konstrukce

```
function plugin0()
{
  return document.getElementById('plugin0');
}
plugin = plugin0;
```

Na plugin lze potom odkazovat zápisem `plugin()`. Možnosti interakce s pluginem jsou následující.

- Volání funkce doplňku s možností uchovat návratovou hodnotu v proměnné. Volané funkci lze předat podle její definice argumenty

  ```
  var ret = plugin().Foo();
  ```

- Zjištění nebo nastavení Property (vlastnosti) pluginu

  ```
  var version = plugin().version;
  ```

- Zachytávat události generované pluginem a reagovat na ně obslužnou funkcí

  ```
  addEvent(plugin(), 'fired', function(x){
  alert('onfired() from plugin : '+x)});
  ```

3.5.9 Debugging

Díky tomu, že plugin běží jako samostatný proces, lze jej snadno odlaďovat připojením debuggeru. V takovém případě je třeba do adresáře pluginu zkopírovat soubor s debugovacími symboly, případně na tyto soubory odkázat v konfiguraci debuggeru. V prostředí Windows se proces jmenuje `plugin-container.exe`, stejné jméno ovšem mohou mít i zcela jiné běžící pluginy.

3.6 Libpurple

3.6.1 Obecný popis

Libpurple je open-source knihovna, která je schopna tvořit jádro instant messaging programu. Je navržena tak, aby se starala o veškerou síťovou komunikaci a správu uživatelských účtů. Vývojáři tedy stačí jednoduše implementovat grafické rozhraní, které bude volat knihovní funkce a obsluhovat události (eventy) generované knihovnou. Libpurple je využívána jako jádro řadou IM programů, jako například Pidgin [47], Instantbird, Finch, Meebo a další. Domovskou stránkou je [42], lze tam nalézt základní příklad použití knihovny a několik článků. Stránka je však velmi málo aktualizována. Lepším zdrojem informací jsou stránky pro vývojáře programu Pidgin [48]. Je zde k dispozici zdrojový kód knihovny jako součást zdrojových kódů Pidginu. Právě tyto kódy jsou jedinou cestou jak zjistit bližší informace o libpurple, podrobnější psané dokumentace není mnoho ani zde. Je zde alespoň komentovaná dokumentace vygenerovaná ze zdrojových kódů [51], která může posloužit jako zdroj programátorských informací.

3.6.2 Technický popis

3.6.2.1 Zdrojový kód

Libpurple je napsaná v jazyce C. Zdrojový kód je koncipován jako multiplatformní a je připraven k buildování pod Windows i Linuxem. Funkce jsou přehledně pojmenovány podle kategorie svého účelu a jsou tak i seskupeny do jednotlivých souborů. Například funkce týkající se práce s lokálními uživatelskými účty jsou definovány a implementovány v souborech `account.h` a `account.c` a jejich názvy začínají prefixem `purple_account_`.

3.6.2.2 Pluginy a protokoly

Možnosti knihovny mohou být rozšiřovány jejími pluginy. Jedná se pouze o pluginy vlastní architektury libpurple, nikoliv o (NPAPI) pluginy do prohlížeče. Budeme je proto raději označovat jako *purple pluginy*. Jejich pomocí lze například realizovat rozšířené možnosti formátování zpráv, uchovávání logu apod. Fyzicky jsou realizovány jako dynamické knihovny (`.dll`, `.so`) a jsou knihovnou načítány při inicializaci, typicky z podadresáře `plugins` v pracovním adresáři knihovny. Analogickým způsobem jsou implementovány některé protokoly s tím rozdílem, že jejich knihovny jsou načítány z pracovního adresáře. Tyto mechanismy budou podrobně popsány v implementační části práce, neboť vyžadují menší úpravy zejména v závislosti na operačním systému.

3.6.2.3 Uživatelské účty

Uživatelské účty a jejich nastavení si knihovna uchovává ve formátu XML v instalačním adresáři. Název souboru je `accounts.xml`. Libpurple umožňuje mít v jedné své instanci uloženo a selektivně připojováno vice účtů na různých protokolech.

3.6.2.4 Main Event Loop

Pro obsluhu příchozích událostí i zpracování lokálních požadavků se v uživatelském rozhraní používá smyčka událostí Glib Main Event Loop [40]. Existují i jiné alternativy, v této práci ale nebudou použity. Smyčka mimo jiné umožňuje definovat časový interval pravidelného volání zvolené callback funkce. Tohoto mechanismu bude možné využít například pro pravidelně vybírání a odbavování fronty zpráv určených odeslání.

3.6.2.5 Thread safety

Knihovna bohužel není thread-safe. To znamená, že ze strany uživatelského rozhraní musí být celou dobu obsluhována výhradně jedním threadem. V opačném případě by docházelo k nepredikovatelným problémům nebo chybám. V implementační části proto bude věnována pozornost dodržení tohoto požadavku.

3.6.3 Kompilace

Zdrojové kódy libpurple jsou součástí zdrojových kódů Pidginu. Vzhledem ke zmíněnému faktu, že kód Pidginu je dobrý zdroj informací o praktickém použití libpurple, rozbalíme pro kompilaci libpurple celý balík zdrojových kódů Pidginu. Po provedení základní konfigurace samozřejmě nebude nutné kvůli buildu libpurple buildovat kompletně celý Pidgin. Kód je sestaven tak, že umožňuje samostatný build libpurple přes `Makefile` z adresáře `libpurple`.

Způsob kompilace je podrobně popsán na stránkách Pidginu obecně a pro Linux [49] i pro Windows [50]. V Linuxu probíhá standardní procedurou `./configure`,`make` a `make install`. Ve Windows se kompiluje pomocí `Makefile` a Mingw kompilátoru. V dalším popisu kompilace se vzhledem ke kvalitní dokumentaci soustředíme pouze na nestandardní nebo nepopsané aspekty.

Pokud není při kompilaci požadována kompletní funkcionalita Pidginu, ale postačuje pouze knihovna libpurple, lze při `./configure` zakázat podporu některých funkcí, pro které tak nebude třeba instalovat závislé knihovny. Zařídit to lze parametrem konfigurace např. `-disable-avahi`.

3.6.3.1 Windows global.mak

Při kompilaci pod Windows se může stát, že budou nesprávně nainstalovány nebo detekovány závislé knihovny, respektive jejich headery. V takovém případě je možno zkontrolovat cesty v souboru `libpurple/win32/global.mak` a případně je ručně upravit

3.6.3.2 Výběr protokolů

Může se stát, že z nějakého důvodu nebude potřeba nebo nebude možno kompilovat podporu určitých protokolů. Například u protokolu Bonjour, protože stažení Bonjour SDK vyžaduje kromě registrace i splnění licenčních podmínek pro vývojáře, viz [37]. V prostředí Windows se neprovádí krok `./configure`, kde by se dala podpora daných protokolů vypnout. Je však možno adresáře příslušných protokolů odstranit z proměnné `SUBDIRS` v souboru `libpurple\protocols\Makefile.mingw`.

3.6.3.3 Podpora SSL v Linuxu

Z licenčních důvodů není možné použít pro podporu SSL knihovnu OpenSSL. Pokud je podpora SSL požadována, je nutno použít buď GNUTLS nebo Mozilla NSS a NSPR. Toto je třeba nastavit při provádění `./configure` procedury. Podrobný popis pro jednotlivé distribuce je na [26].

3.6.3.4 Statické linkování

Ve Windows je implicitně nastavena tvorba dynamicky linkované knihovny. V `Makefile.mingw` souboru v adresáři `libpurple` je v příslušné sekci nastaveno statické linkování.

```
$(TARGET).dll $(TARGET).dll.a: $(OBJECTS)
 $(CC) -shared $(OBJECTS) $(LIB_PATHS) \$(LIBS)
 $(DLL_LD_FLAGS) -Wl,--output-def,$(TARGET).def,
 --out-implib,$(TARGET).dll.a -o $(TARGET).dll
```

Pokud budeme požadovat staticky linkovanou knihovnu, je nutno změnit nebo doplnit příkaz a použít nástroje `ar`. Příklad použití je

```
$(AR) cru $(TARGET).lib $(OBJECTS)
```

Záměr vytvořit ve Windows pomocí statických knihoven jediný soubor prohlížečového pluginu, který by obsahoval všechny svoje závislosti v sobě, se však nepodařilo zrealizovat. Problémy nastávaly zejména se statickým linkováním purple pluginů. Protože libpurple stejně potřebuje uchovávat v pracovním adresáři další soubory, například nastavení uživatelských účtů a certifikátů, bylo od vize pluginu v jediném souboru upuštěno.

V Linux naopak staticky linkované protokoly využít lze. Ušetří to část problémů s hledáním příslušných knihoven. Výběr statických protokolů se provádí při `./configure` pomocí parametru např. `-with-static-prpls=oscar`.

3.6.4 Ukázková aplikace

Při tvorbě prvotní pokusné aplikace pro příkazovou řádku je možné použít kód, který je přímo v balíčku v souboru `libpurple\example\nullclient.c`. Další příklady kódu jsou na [42]. Příklady jsou poměrně dobře komentovány. Zprovozněná ukázková konzolové aplikace se nachází i na přiloženém CD.

3.6.4.1 Nastavení parametrů spojení

Ukázkový příklad na přiloženém CD obsahuje v kódu možnost nastavení parametrů spojení pro případ změny přihlašovacího serveru a typu zabezpečení nebo připojování přes proxy server. Tato nastavení v ukázkových příkladech nejsou.

Parametry spojení se nastavují jako vlastnosti daného uživatelského účtu. Uchovávány jsou ve formě dvojic jmen a hodnot, kde typ hodnoty koresponduje s typem v nazvu funkce `purple_account_set_TYP`. Nastavené hodnoty by se měly uložit i do souboru `accounts.xml` s uživatelskými účty.

```
purple_account_set_int(account, "port", 443);
purple_account_set_string(account, "server", "login.icq.com");
```

```
purple_account_set_string(account, "connection_security", "none");
purple_account_set_bool(account, "require_tls", false);
purple_account_set_bool(account, "old_ssl", TRUE);
purple_account_set_bool(account,"use_clientlogin", FALSE);
```

Nastavení proxy serveru se provádí následujícím způsobem. Typy podporovaných proxy serverů jsou definovány v souboru `proxy.h` v enumu `PurpleProxyType`. Pokud spojení nepoužívá proxy, je možné nastavit hodnotu `proxy_info` na `NULL`.

```
PurpleProxyInfo *proxy_info = purple_proxy_info_new();
purple_account_set_proxy_info(account, proxy_info);
purple_proxy_info_set_type(proxy_info, PURPLE_PROXY_SOCKS5);
purple_proxy_info_set_host(proxy_info, "proxyserver");
purple_proxy_info_set_port(proxy_info, 8080);
```

3.7 Libjingle

3.7.1 Obecný popis

Libjingle je open-source knihovna, která umožňuje tvorbu peer-to-peer aplikací. Stará se o vytvoření spojení, vyjednání jeho parametrů a samotný přenos dat. Umí navazovat spojení i přes NAT a firewall. Dále umí komunikovat pomocí XMPP protokolu [56]. Je určena zejména jako jádro aplikací pro hlasový a video chat nebo přenos souborů [5]. Je vyvíjena společností Google a je vydána pod licencí Berkeley-style license [3]. Ta umožňuje její použití a distribuci v komerčním i nekomerčním software.

Cílem této práce bude zmapovat a případně implementovat funkcionalitu použitelnou na hlasovou a video komunikaci s Google chatem. K tomu poslouží jeden z ukázkových programů distribuovaných přímo ve zdrojovém balíčku – `call`. Ten umožňuje přihlášení ke Google chatu, získání kontaktů (rosteru) a zahájení hlasového nebo video hovoru. Bohužel tento příklad podporuje pouze přenos ukázkových audio a video dat uložených v dump souboru. Bude tedy nutné prozkoumat možnosti přenosu živého audia a video snímaného ze zachytávacích karet.

3.7.2 Technický popis

Libjingle je psaná v jazyce C++. Zdrojový kód je dostupný na Google Code stránce [9]. Zde jsou dispozici jsou připravené balíčky jednotlivých verzí a Subversion repozitář s nejnovější verzí kódu. Balíčky jsou proti repozitáři mírně neaktuální, ovšem kód stažený přímo z repozitáře někdy vyžaduje drobné úpravy aby byl kompilovatelný. Je zde i Issues fórum pro řešení konkrétním problému v rámci komunity uživatelů. Vývojářská dokumentace je na velmi dobré úrovni, na [5] jsou v Developer Guide popsány základní principy a technologie včetně názorných grafických schémat [8]. Stručně proto popíšeme pouze nejdůležitější z nich.

3.7.2.1 Thread safety

Knihovna podporuje multithreading v tom smyslu, že umožňuje rozdělení své činnosti do dvou základních threadů.

- Signaling thread pro vytváření a obsluhu řídících struktur
- Worker thread (také nazývaný Channel thread) využívaný peer-to-peer komponentami k realizaci úloh náročných na systémové prostředky, například streamování dat.

Tím se zamezí vzájemnému blokování XMPP zpráv a uživatelského rozhraní s dlouhými přenosy dat. Nedá se ovšem spolehnout na to, že všechny metody jsou thread-safe. Pouze některé z nich zamykají kritické sekce nebo provádějí kontrolu volajícího threadu. Celý koncept threadingu je velmi podrobně popsán v příslušné kapitole [8].

3.7.2.2 Spojení

Mechanismus spojení se skládá ze dvou kanálů.

- Session negotiation channel (také zvaný signaling channel) zakládá, udržuje a ruší spojení. Dále vyjednává a nastavuje jeho parametry, například potřebné kodeky pro přenos obsahu. To se děje pomocí XMPP zpráv.
- Data channel zabezbečuje přenos samotných dat, například videa. V závislosti na vyjednaných parametrech se používá TCP nebo UDP protokolu. Důležitým faktem je, že tento kanál neprobíhá přes XMPP protokol.

Pro průchod firewallem, NATem apod. se používá technologie STUN [34]. Je vytvořena skupina spojení různého typu a parametrů a je sledována jejich kvalita. Postupně se z této skupiny eliminují nevyhovující spojení a používá se to aktuálně nejvýhodnější.

3.7.2.3 Google Talk rozšíření XMPP

Google Talk používá několik nestandardizovaných rozšíření protokolu XMPP [4] [29]. Jsou označovány jako protokol Jingle. Definovat je umožňuje koncept rozšiřitelnost tohoto protokolu. Jejich podoba však zatím není definitivní a v průběhu vývoje se může měnit. To může být příčinou některých problémů, které budou diskutovány v implementační části.

3.7.3 Třídy MediaEngine

Třída MediaEngine [10] je nejnižší vrstvou knihovny libjingle. Tvoří wrapper nad vybranou knihovnou třetí strany, která provádí zachytávání a zobrazování audia nebo videa ze zachytávacích karet. Třídy od ní odvozené, založené na různých technologiích třetích stran, tak mají jednotné rozhraní. MediaEngine může obsahovat audio (VoiceChannel) i video kanál (VideoChannel), oba zároveň nebo jeden z nich. Každý z nich může být schopen příjmu a vysílání (opět buď jednoho z toho nebo obojího).

3.7.3.1 Třída VoiceChannel

Třída VoiceChannel [12] posílá zvuková data ze zachytávacího zařízení do transportního kanálu ve formě paketů. Rovněž provádí obrácený proces, kdy přijímané pakety přehrává na zvukové kartě. Její rozhraní umožňuje přerušit přenos nebo jej ztišit (mute). Dvě velmi důležité metody jsou

- `MediaChannel::OnPacketReceived` která přijímá a zpracovává data z transportního kanálu. Zpracování obvykle znamená prosté přeposlání knihovně třetí strany, která pakety rozkóduje a přehraje.
- `VoiceChannel::SendPacket` slouží k odesílání odchozích dat transportním kanálem

Zcela analogicky funguje i VideoChannel zajišťující přenos video obsahu.

Bylo vytvořeno více implementací MediaEngine založených na různých technologiích. Některé z nich najdeme přímo ve zdrojových kódech. Většina z nich je tam ovšem pouze ve formě nikde nepoužitého zdrojového souboru, který navíc často ani není delší dobu udržován a vyžaduje úpravy, aby byl alespoň zkompilovatelný.

3.7.3.2 FileMediaEngine

Jde o engine použitý v ukázkovém programu `call`. Nezachytává živý zvuk a obraz, používá pouze krátké ukázkové sekvence uložené v souborech. Odtud vychází pojmenování enginu. Zmíněné soubory jsou pouze pakety uložené v binární formě. Postupně se načítají a odesílají, čímž je simulováno snímání a streamování média. Opačný postup lze použít pro zápis do dump souboru.

3.7.3.3 WebRtcMediaEngine

Tento engine je založený na projektu WebRTC [55]. Ve zdrojových kódech je přiložen, ale jeho třída není použita. Vzhledem k tomu, že vývoj WebRTC je v současné době podle frekvence prováděných změn velmi intenzivní [39], nabízí se možnost navázání na tuto práci tímto směrem.

3.7.3.4 LinphoneMediaEngine

LinphoneMediaEngine je založen na jádru aplikace Linphone [43], které je tvořeno knihovnou Mediastreamer2 [44]. Obecnému popisu těchto technologií je věnována kapitola 3.7.4.2. Ve zdrojových kódech je stará nezkompilovatelné verze. Podle příspěvků na fóru [9] je však po úpravách použitelná na přenos živého audio a video streamu. Úprava toto mediaenginu pro použití s aktuální verzí libjingle a zprovoznění streamování živého audia a videa bude cílem této práce.

Na vývojářských fórech (např. [9]) lze najít řešení téhož problému v různém stupni kompletnosti a kvality. Kód je ve všech případech velmi podobný, ovšem žádné všeobecně uznávané kompletně funkční řešení se nalézt nepodařilo. V implementační části práce budou poznatky z jednotlivých diskusí analyzovány a implementovány do řešení této práce. Do budoucna bude důležité neustále sledovat uvedené zdroje těchto poznatků, protože příslušné diskuse jsou většinou aktivní a kdykoli se mohou objevit nové důležité informace.

3.7.4 Související software a knihovny

3.7.4.1 Protokoly oRTP a SRTP

Protokol oRTP (Real-time Transport Protocol) [46] je protokol používaný pro streamování (nejen) mediálních dat v reálném čase. Jeho hlavním úkolem je zajišťovat časovou synchro-

nizaci přenosu. SRTP (Secure Real-time Transport Protocol) [33] protokol umožňuje zabezpečení přenášených dat. RTCP (RTP Control Protocol) [32] je protokol přenášející kontrolní a statistické a QoS informace RTP přenosů. Typicky se pro RTP přenos používá port se sudým číslem a pro RTCP port s číslem o jednotku vyšším (lichým).

3.7.4.2 Linphone

Linphone [43] je open-source VoIP software, který umožňuje audio, video a textovou komunikaci. Jako svůj hlavní protokol používá SIP. Navzdory svému názvu je dostupná pro Windows, Linux i Mac a navíc i pro mobilní platformy Android, iPhone a Blackberry. Z tohoto pohledu se její použití v této práci jeví jako velmi výhodné, i s výhledem na možnou budoucí implementaci na některé mobilní platformě. Hlavní součástí aplikace, která bude využita v této práci, je knihovna Mediastreamer2.

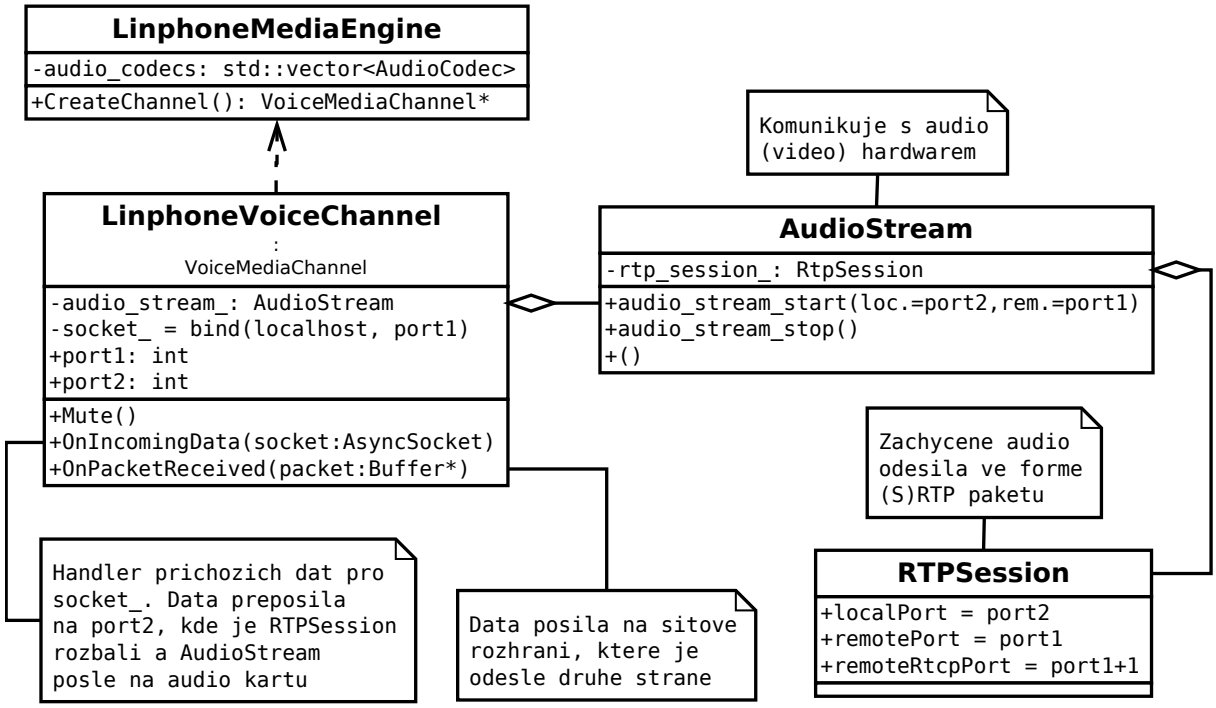
3.7.4.3 Mediastreamer2

Klíčovou knihovnou zajišťující v Linphone streaming živého audia a videa ze zachytávacích karet je Mediastreamer2 [44]. Je psán v jazyce C a je licencován jako GPL. Obstarává kompletně zachytávání, kompresi, dekompresi, přenos a zobrazení multimediálního obsahu. Podporuje řadu kodeků pro audio (speex, G711, GSM, iLBC, AMR , SILK, G729) a video kompresi (H263, theora, MPEG4, H264 a VP8). Umožňuje i zápis nebo čtení `.wav` souborů. Do procesu streamování umožňuje vkládat různé filtry, například na potlačování ozvěny. Jeho funkce lze rozšířit pomocí pluginů.

Způsob kompilace je dobře zdokumentován. Pro získání knihoven Mediastreameru2 opět není potřeba buildovat celý Linphone. Jeho zdrojový kód je navíc k dispozici i samostatně. Samotný kód Linphone využit nebude. Je však potřeba povolit při konfiguraci podporu videa `-enable-video` a pro podporované kodeky mít v systému nainstalované jejich knihovny a headery (`ffmpeg`).

Mediastreamer2 má struktury `AudioStream` a `VideoStream` reprezentující kanál s mediálními daty a sadu obslužných funkcí. Způsob práce s oběma z nich je zcela analogický, popsán bude na příkladu `AudioStream`. Všechny funkce s ním související mají prefix `audio_stream_`. Pracuje tak, že mediální data zachycená ze zařízení zpracuje definovanými filtry, zakóduje pomocí vybraného kodeku a odešle přes protokol RTP, případně jeho zabezpečenou verzi. Proces příjmu je obrácený s tím, že končí přehráním nebo zobrazením média na kartě. Za těmito účely se otvírá při vytvoření AudioStreamu naslouchací port pro příjem dat. Při jeho spouštění se specifikuje vzdálená adresa a dvojice portů (pro data a řídící RTCP signalizaci).

Pro práci s AudioStreamem jsou nejdůležitější dvě funkce. `audio_stream_new(int locport, bool_t ipv6)` je funkce, která vrací novou instanci AudioStreamu se specifikovaným naslouchacím lokálním portem `locport`. Samotný přenos dat je třeba zahájit některou z variant funkce `audio_stream_start_`, jejíž nejdůležitější parametry jsou `remip` (vzdálená adresa kam se budou posílat data), `remport`(port pro odesílání dat), `rem_rtcp_port`(řídící RTCP port), `payload`(specifikace formátu kódování obsahu). Vše je podrobně zdokumentováno na [45] a znázorněno na obrázku 3.1.



Obrázek 3.1: Struktura LinphoneMediaEngine a VoiceChannelu

3.7.5 Kompilace

Kompilace libjingle je velmi podrobně popsána v souboru `README` ve zdrojových kódech. Probíhá pomocí frameworku SCons, což je sada pythonových skriptů volajících podle nastavených parametrů kompilátor a linker. Ve Windows se žádný projekt pro Visual Studio nevytváří. Produktem kompilace jsou statické knihovny a ukázkové aplikace. Funkčnost zkompilovaného produktu lze vyzkoušet například ukázkovým programem `call`.

V případě chybového hlášení skriptu `site_init.py` ohledn2 nestringov0ho typu path pomohly změny na těchto řádcích.

```
330: SCons.Script.Main._load_site_scons_dir(
    str(SCons.Node.FS.get_default_fs().SConstruct_dir), site_dir)
450: SCons.Script.Main._load_site_scons_dir(
    str(SCons.Node.FS.get_default_fs().SConstruct_dir), None)
```

Kapitola 4

Návrh a implementace řešení

Tato kapitola navrhne obecný postup tvorby pluginu a jeho uživatelského rozhraní. Podle něho bude v následujících dvou podkapitolách popsán návrh a implementace obou pluginů. Důraz bude kladen na ty aspekty implementace, které jsou vlastním přínosem této práce a nejsou nikde veřejně popsány.

4.1 Obecný postup práce

Cílem práce je vytvořit pluginy do prohlížeče volající funkce knihoven libpurple a libjingle z HTML a JavaScriptu. Obecný postup práce u obou knihoven bude následující.

- Instalace a konfigurace nástrojů potřebných pro build a vývoj, nastavení prostředí
- Build knihovny a ukázkových konzolových aplikací z výchozího zdrojového kódu
- Implementace nebo úprava kódu ukázkových aplikací a knihoven tak, aby konzolová aplikace byla schopna přihlášení se ke službě a odesílání a příjmu
- Vytvoření PurpleBreath pluginu propojeného s tímto dopracovaným kódem
- Tvorba uživatelského rozhraní v HTML a JavaScriptového API pluginu
- Testování funkčnosti řešení

4.2 Libpurple plugin

4.2.1 Konzolová aplikace

4.2.1.1 Build

Build knihovny libpurple proběhl podle postupu popsaného v kapitole 3.6.3. Výstupem byla dynamicky linkovaná knihovna `libpurple.dll` a import knihovna libpurple.dll.a.

Následoval build ukázkové aplikace s kódem podle [42]. Pro Linux jsem použil jednoduchý Makefile projekt, pro Windows jsem zvolil Visual Studio. I pod Windows však lze použít Makefile projekt pro Mingw kompilátor. V projektu je nutno nastavit cesty ke hlavičkovým souborům `.h` a cestu k linkovaným knihovnám pro libpurple a glib. Knihovna glib je jednou

ze závislostí potřebných pro kompilaci Pidginu (libpurple), nachází se tedy v příslušném podadresáři build projetu Pidginu. Vzhledem k možnosti existence jiných verzí této knihovny ve standardních include adresářích bylo zvoleno nastavení právě do projektu Pidginu.

Do výstupního adresáře je potřeba nakopírovat všechny knihovny, na kterých libpurple závisí a nejsou nainstalované v systému. Závislosti knihovny lze ověřit pomocí programu dependency Walker nebo `ldd`. Kromě toho je nutné do výstupního adresáře nakopírovat knihovny purple protokolů a do podadresáře `plugins` purple pluginy rozšiřující funkcionalitu a podporovaná protokoly. Není potřeba kopírovat pluginy, jejichž funkcionalita není vyžadována. Dále je do výstupního adresáře nutné nakopírovat certifikáty certifikačních autorit, které mohou být potřeba pro zabezpečené připojení k serveru. Zkopírujeme celý adresář `ca-certs` do výstupního adresáře projektu (ve výstupním adresáři tedy bude podadresář `ca-certs`).

Projekt s konzolovou testovací aplikací je umístěn na přiloženém CD v adresáři `PurpleSampleClient`

4.2.1.2 Test funkčnosti

Nyní by měla být ukázková aplikace spustitelná a měla by být schopna připojit se ke zvolené IM službě a přijímat textové zprávy. Program produkuje poměrně obsáhlý textový výstup o svojí činnosti, z něho by mělo být patrné k jaké chybě případně došlo. Rozšířený výpis ladicích zpráv lze nastavit pomocí funkce `purple_debug_set_enabled`. Při testu připojení ke službě ICQ ve standardním nastavení přenosových parametrů by mělo dojít k inicializaci SSL, stažení certifikátu přihlašovacího serveru do podadresáře `certificates`. Tento podadresář nemusí být při spouštění přítomen, knihovna si jej v případě potřeby založí a certifikát stáhne.

4.2.1.3 Diagnostika problémů

Při diagnostice případného problému se spojením je kromě textového výstupu programu užitečné a rychlé přesvědčit se, že program dokáže stáhnout certifikát přihlašovacího serveru. Pokud ne, pravděpodobně je problém již v inicializaci systému SSL, například chybí potřebná knihovna. V takové situaci lze vyzkoušet nezabezpečené přihlášení k serveru, jehož parametry se nastavují způsobem popsaným v 3.6.4.1.

Dalším souborem který aplikace zakládá v pracovním adresáři je `accounts.xml`, kam se ukládají uživatelské účty. Kontrola přítomnosti a obsahu tohoto souboru může být dalším vodítkem při odlaďování problémů. Může se stát, že aplikace zapisuje tento soubor do jiného adresáře než očekáváme, což značí problémy s nastavením cest. Tyto problémy budou podrobně diskutovány při implementaci finálního pluginu.

Dalším problémem mohou být restriktivní pravidla firewallu nebo proxy server. Vlastním nastavením parametru připojení podle 3.6.4.1 lze nastavit použití proxy serveru. Kontrolu spojení na nejnižší úrovni lze provést pomocí software zachycujícího síťový provoz, například programu Wireshark.

4.2.2 Vytvoření FireBreath pluginu

Poté, co byla ověřena funkčnost konzolové aplikace, můžeme přistoupit ke tvorbě FireBreath pluginu. Název libpurple pluginu byl v této zvolen *PurpleBreath*. Nyní bude třeba nastavit

linkování knihovny libpurple k projektu pluginu a vytvořit API pro komunikaci s JavaScriptem.

4.2.2.1 Vytvoření a nastavení projektu

Vytvoříme nový projekt podle popisu v 3.5.3. Nejprve je třeba upravit obecné i platformní konfigurační soubory generátoru projektu. Ve zdrojovém adresáři projektu vytvoříme podadresář `deps`. Do jeho podadresáře `include` umístíme headery knihoven libpurple a glib. Do druhého podadresáře `lib` potom knihovny pro všechny podporované operační systémy. Takto zvolené umístění umožní odkazovat relativní cestou stejným způsobem ve všech platformách správné verze a dobrou možnost archivace a přenesení projektu na jednom místě. Alternativou samozřejmě může být použití systémového mechanismu pro include a linkování knihoven, je ale potřeba dbát na nainstalování jejich správné verze. Ve společném `CMakeLists.txt` nyní doplníme include adresáře, tyto změny se budou při generování promítat do výsledných projektů.

```
include_directories(
  ${PLUGIN_INCLUDE_DIRS}
  "${CMAKE_CURRENT_SOURCE_DIR}/deps/include/purple"
  "${CMAKE_CURRENT_SOURCE_DIR}/deps/include/glib"
  )
```

Obdobným způsobem, ale zvlášť pro každou platformu, nastavíme i linkované knihovny. Tím je připravena základní infrastruktura projektu. V Linuxu lze ještě nastavit parametr linkeru `rpath` podle popisu v následující kapitole. Poté je možno vygenerovat buildovací projekt a vyzkoušet samotné provedení buildu.

4.2.2.2 Nastavení rpath v Linuxu

V Linuxu je možné nastavit parametr linkeru `rpath`, který bude zajišťovat, že závislé knihovny pluginu budou přednostně hledány v tom adresáři, kde se nachází samotná knihovna pluginu. To umožní distribuovat plugin i potřebnými knihovnami a nevyžadovat po uživateli jejich zvláštní instalaci. Navíc to pomáhá řešit problém s vyhledáváním purple pluginů, s čímž může mít proces FireBreath doplňku potíže protože je spouštěn z jiného adresáře. Toto nastavení a jeho vhodnost nebo nevhodnost bude zřejmě třeba zvážit v případě problémů na konkrétní platformě.

Nastavení se provádí ve vygenerovaném souboru `link.txt` jako první parametr ve tvaru `-Wl,-rpath,$ORIGIN`. Je to nesystémové řešení, protože se ztrácí při přegenerování projektu. Bohužel se zatím nepodařilo nalézt lepší funkční způsob.

4.2.2.3 Úprava proměnné PATH ve Windows

Ve Windows bylo potřeba upravit proměnnou PATH procesu pluginu. Protože je plugin spouštěn z instalačního adresáře prohlížeče v plugin-container procesu, nastávaly by jinak problémy s vyhledáváním závislých knihoven a purple pluginů. Děje se tak v API metodě `Init` pomocí funkce `_putenv_s`.

4.2.3 Tvorba JavaScriptového API

Poslední fází tvorby pluginu bude návrh a implementace API pro komunikaci s JavaScriptem. Obecné principy tvorby rozhraní byly popsány v 3.5.4. Konkrétní implementace vyjde z odzkoušeného kódu ukázkové konzolové aplikace. Kód API bude obsahovat jeho bloky téměř beze změny. Při implementaci dalších funkcí, které v ukázce nejsou, budeme vycházet ze způsobu jejich implementace v kódu Pidginu. Jak již bylo uvedeno, kód libpurple i Pidginu je přehledně členěn tak, že není problém intuitivně dohledat požadovanou funkci. Metody API budou většinou obsahovat pouze kontrolu vstupních parametrů a volání knihovní funkce.

4.2.3.1 EventLoop thread

Zásadním rozdílem fungování pluginu proti konzolové aplikaci je to, že konzolová aplikace používá nekonečnou smyčku Main Event Loop, která zajišťuje obsluhu událostí a výpisy na konzoli. Tímto způsobem plugin nemůže fungovat, protože by se po zavolání příslušné funkce JavaScriptem nikdy nevrátilo řízení zpět a celé okno prohlížeče by přestalo reagovat. Tento problém byl vyřešen tak, že jádro libpurple je uvnitř pluginu spuštěno ve zvláštním threadu. Funkce volaná z JavaScriptu jej pouze nastartuje a vrátí se zpět. Protože knihovna není thread-safe, je třeba počítat s možnými problémy.

Jedno z možných řešení, použité pro odesílání zpráv, je ukládat odesílané zprávy do fronty a zaregistrovat v hlavní smyčce callbak t vhodným timeoutem, který je bude vybírat a odesílat. Toto je implementováno v souboru `PurpleDefines.h` pomocí struktury `SendMsgQueueItem` a funkce `enqueue_msg`. Každých 500ms se volá v rámci hlavní smyčky callback zaregistrovaný pomocí
`g_timeout_add (500 , sender_timeout_callback , m_gmainloop);`
který z fronty odebere a odešle všechny čekající zprávy.

Alternativou k tomuto řešení by mohlo být použití signálu. Pidgin, jehož uživatelské rozhraní je psáno v GTK, spoléhá na jeho implementaci hlavní smyčky a thread-safe volání metod [41].

4.2.3.2 Příklad vytvoření API funkce

Vytvoření API funkce demonstrujeme na příkladě funkce, která maže uživatelský účet. Snadno dohledáme, že funkcí k tomu určenou je `purple_accounts_delete(PurpleAccount * acc)`. Její argument, tedy struktura reprezentující daný účet, se získá vyhledáním pomocí funkce `purple_accounts_find`, která hledá účet specifikovaný uživatelským jménem a názvem protokolu. Pokud bylo hledání úspěšné, můžeme nalezený účet smazat. Celá metoda je implementována takto.

```
void PurpleBreathAPI::DeleteAccount(string name, string proto)
{
  PurpleAccount * acc;
  if(!(acc = purple_accounts_find(name.c_str(), proto.c_str())))
   return;

  purple_accounts_delete(acc);
}
```

Zbývá ji zaregistrovat v konstruktoru pomocí

```
registerMethod("DeleteAccount",
                    make_method(this, &PurpleBreathAPI::DeleteAccount));
```

Seznam a popis funkcí API rozhraní je v tabulce C.

4.2.4 Tvorba GUI

Posledním krokem bude vytvoření ukázkového uživatelského rozhraní. Bude tvořeno jednou HTML stránkou a sadou JavaScriptových funkcí. Jeho kód nebude obsahovat nikterak složitou logiku, většinou pouze zkontroluje vstupní data z formulářů a zavolá APi funkci. Při jeho implementaci je třeba mít na zřeteli následující aspekty.

- Pro inicializaci libpurple je třeba při startu aplikace zavolat funkci `Init()`. Tato funkce se volá automaticky a měla by se volat pouze jednou.
- Číselné označení jednotlivých typů proxy musí zůstat zachováno tak, aby odpovídalo značení definovanému v knihovně enumem `PurpleProxyType`.
- Není podporováno spouštění více instancí pluginu. I pro více účtů na různých protokolech však postačí jedna instance.

Rozhraní je schopno vytvoření, ukládání a rušení účtů. Umožňuje se ke službě přihlásit a zobrazit seznam kontaktů. Do něho lze přidávat i rušit kontakty. Dále zobrazuje přijaté zprávy a umí vybranému kontaktu zprávu napsat. Zobrazuje okno se záznamem o důležitých událostech, zejména chybách. Podrobný popis je v uživatelské příručce v kapitole B.1.

4.3 Libjingle plugin

Nejprve bude třeba zaktualizovat a doimplementovat třídu `LinphoneMediaEngine`. Její funkčnost budeme poté ověřovat ukázkovým programem `call` upraveným pro použití LinphoneMediaEngine.

4.3.1 Úprava implementace LinphoneMediaEngine

4.3.1.1 Kompilace

Rozdíl proti kompilaci podle popisu v kapitole 3.7.5 bude v tom, že přidáme do souboru `libjingle.scons` do sekce `libjingle cppdefines` definice `HAVE_LINPHONE` a do `srcs` zdrojový soubor `session/phone/linphonemediaengine.cc`. Dále nastavíme potřebné include adresáře a knihovny. Hotová verze je v projektu na přiloženém CD. Tamtéž najdeme i upravené soubory týkající se LinphoneMediaEngine. Jsou to zejména `linphonemediaengine.h` a `.cc` a `mediaenginefactory.h` a `.cc`. Provedené změny jsou popsány v následujících podkapitolách. Výstupem kompilace budou statické knihovny (narozdíl od libpurple opravdu použitelné jako statické) a ukázkové příklady aplikací.

4.3.1.2 Zdroje implementačních úprav

Je velmi doporučeno neustále sledovat aktuální stav uvedených zdrojů. Většina z nich je aktivních a mohou přinést nové důležité informace.

Nejucelenějším zdrojem úprav audio části enginu je [28]. Velkým přínosem je implementace dynamického přidělování portů a runtime detekce audio knihoven. Zatím však neřeší přenos videa. Přenosem videa se zabývá issue [6], kde je od přispěvatelů několik návrhů implementace video části enginu. Sám jsem do diskuse přispěl při řešení menšího problému s kapacitou bufferu packetu [7]. Rovněž jsme si mailovou cestou vyměňovali poznatky s autorem tohoto issue.

4.3.1.3 Dynamické přidělování portů

V originální implementaci jsou lokální porty zadány napevno. Dynamický způsob nechává na systému, aby našel a použil nějaký volný port. Takto získané porty se uloží ihned po přidělení do členské proměnné. Fragment kódu a komentáře popisuje způsob jak toto provést pro socket a lokální port u RTP session. Analogického postupu lze použít i u video streamu.

```
/* 0 means that OS will choose some free port */
socket_->Bind(talk_base::SocketAddress("localhost", 0));

/* and here we get port choosed by OS */
port1 = socket_->GetLocalAddress().port();

/* -1 means that function will choose some free port */
audio_stream_ = audio_stream_start(&av_profile, -1,
    "localhost",port1, i->id, 250, 0);
port2 = rtp_session_get_local_port(audio_stream_->session);
```

4.3.1.4 Kodeky a Payload

Payload type určuje, v jakém formátu (kodeku) se budou data přenášet [53]. Tento parametr se nastavuje při spouštění streamu. Je potřeba zvolit takový payload, aby příslušný kodek podporovaly obě strany přenosu. Zároveň je třeba nastavit takové rozlišení videa, které kodek podporuje. Seznam kodeků, které podporuje Google Talk, je na [54].

Audio kodeky

- PCMA, PCMU, G.722, GSM, iLBC, Speex

Video kodeky

- H.264/SVC
- H.264
- H.263-1998
- p5ipravuje se podpora kodeku Google VP8

Při testování audia je dobré v případě problému nastavit kodek PCM, což může vyloučit problémy s instalací jiných kodeků. V [28] je navíc implementována detekce přítomných kodeků za běhu aplikace. Pro video se po řadě pokusů osvědčil kodek H.263-1998 při rozlišení 352×288.

4.3.1.5 Mediastreamer zařízení

Při spouštění streamu Mediastreamer2 automaticky detekuje zachytávací zařízení (zvukové karty, videokamery) a automaticky vybírá výchozí. Pozor, nezaměňovat se zařízeními které detekuje vlastním mechanismem libjingle (v příkladu `call` je vypisuje příkazem `getdevs`. V implementaci jsem ponechal detekovanou výchozí kartu. Pokud by bylo potřeba vybrat jinou, je třeba specifikovat ji názvem. Názvy všech nalezených karet lze vypsat, což je v kódu implementováno zpracováním výstupu funkce `ms_snd_card_manager_get_list`. Výběr výchozí zachytávací karty je proveden takto.

```
MSSndCard *captcard = ms_snd_card_manager_get_default_capture_card(m);
```

respektive

```
MSSndCard *captcard = ms_snd_card_manager_get_card(m,"ALSA: Intel ICH5");
```

Pro videokameru je způsob analogický za použití funkcí s prefixem `ms_web_cam_`. V případě problému je ve výpisech aplikace nutno zkontrolovat, jaká karta se pro zachytávání používá, případně nastavit správnou.

4.3.1.6 Nastavení proxy

Pro nastavení proxy serveru jsem vyzkoušel následující úpravy v souboru `call_main.tt` umístěné za vytvoření a nastavení `XmppClientSettings xcs`. Podporované typy proxy jsou definovány ve výčtu `talk_base::ProxyType`.

```
xcs.set_proxy(talk_base::PROXY_SOCKS5);
xcs.set_proxy_host("localhost");
xcs.set_proxy_port(8080);
```

Pokusy s přihlášením přes HTTP nebo SOCKS5 proxy se bohužel přesto nezdařily. Příčinu se zatím nepodařilo odhalit, je možné že to bylo způsobeno nastavením použité sítě, které bylo ohledně přístupu na Internet dost restriktivní.

4.3.2 Test komunikace

Po buildu upravené verze knihovny budeme vycházet z ukázkového programu `call`. Nejprve jej vyzkoušíme a zprovozníme jako konzolovou aplikaci, poté přistoupíme ke tvorbě pluginu. Nezávisle na použitém enginu by mělo fungovat přihlášení k účtu na Google chatu seznam online kontaktů.

4.3.2.1 Test FileMediaEngine

Následující parametry programu call `xcs` nastaví použití FileMediaEngine, která místo ze živých zařízení streamuje obsah specifikovaných dump souborů. Dumpy jsou standardní součástí zdrojového balíčku, jsou použity jejich relativní cesty vzhledem k adresáři s programem call.

```
call.exe --videoinput ../../../session/phone/testdata/video.rtpdump
```

```
--voiceinput ../../../session/phone/testdata/voice.rtpdump
```

Po přihlášení a zahájení hovoru (`call`) nebo videohovoru (`vcall`) se bude streamovat v nekonečné smyčce ukázkový zvuk nebo video.

4.3.2.2 Test LinphoneMediaEngine

Při spuštění programu bez výše uvedených parametrů se pro přenos bude používat LinphoneMediaEngine. Při videohovoru bude MediaStreamer2 vytvářet grafické okno s náhledem vlastní kamery a s přijímaným videem. I přesto, že jde o konzolovou aplikaci, je tedy tedy nutné ji spouštět v grafickém prostředí.

V ideálním případě by měl obousměrně fungovat přenos audia i videa. To se však mě, ani přispěvatelům různých diskusí zcela dokonale nepodařilo. Důležité je vyzkoušet jak spojení aplikace z originálním Google chatem, tak spojení dvou aplikací. Podrobný popis testů a rozbor výsledků je proveden v kapitole 5.3.5

4.3.3 Implementace pluginu

Rovněž bude třeba upravit kód konzolové aplikace tak, aby volané API funkce vracely řízení a nezablokovaly okno prohlížeče. Můžeme přitom využít hotovou třídu `Console`, která načítá vstup konzole a získané příkazy posílá pomocí metody `Console` jako signály klientskému threadu.

Vygenerujeme nový FireBreath projekt. Pro libjingle plugin byl v této práci zvolen název projektu *JingleBreath*. Do podadresáře `deps` ve zdrojovém adresáři umístíme headery a knihovny libjingle, Mediastreamer2 a oRTP. Nastavení include adresářů a linkovaných knihoven bude zcela analogické jako u libpurple (viz kap. 4.2.2.1). Upravené zdrojové kódy vycházející z původní implementace programu `call` jsou budou umístěny v podadresáři `call` ve zdrojovém adresáři projektu.

Úprava třídy `Console` bude spočívat v tom, že v její metodě `RunConsole` odstraníme smyčku která načítá příkazy ze standardního vstupu. Vstup nahradíme voláním nové funkce `PerformCommand`, která pouze provede post klientskému threadu.

Samotná implementace bude provedena stejným jednoduchým způsobem jako u libpurple pluginu. Po kontrole vstupních údajů se příslušný příkaz přes `Console` odešle klientskému threadu. Pro vracení hodnot JavaScriptu je, jako například u získávání seznamu kontaktů, použit `FB::VariantList`. V tomto konkrétním případě lze využít naimplementovaný mechanismus knihovny vracející tento seznam v `std::vector`.

Tabulka s popisem API funkcí je na D.

4.3.4 Uživatelské rozhraní

Opět bylo implementováno pomocí standardních HTML prvků a umožňuje přihlášení, získání seznamu kontaktů a zahájení a přijetí hlasového nebo video hovoru. Instalace a stručný návod k použití je v kapitole B.2.

Kapitola 5

Testování

Tato kapitola popisuje metodiku provedených testů a jejich výsledky pro hlavní funkce obou pluginů a jejich rozhraní.

5.1 Metodika testů

Testování probíhalo interně v rámci komunity vývojářů ExtBrainu. S uvolněním se počítá až po integraci do ExtBrain Communicatoru. Hlavní testovací platformou byl prohlížeč Mozilla Firefox ve verzi 12 v prostředí Windows i Linux. Důraz byl kladen na testování služby ICQ. Testovací metodika byla ověřování hlavní funkcionality metodou Black Box.

Dále jsem PurpleBreath plugin, spolu se stručnou instrukcí jak se přihlásit k ICQ, odeslal dvěma neobeznámeným uživatelům stojícím zcela mimo projekt ExtBrain. Oba pokusy byly úspěšné.

5.2 PurpleBreath plugin

5.2.1 Instalace pluginu

Na všech platformách se podařilo podařilo rozšíření s pluginem bez problému nainstalovat. Testován byl XPI instalátor pro Firefox ve Windows i Linuxu a MSI instalátor pro Internet Explorer. Plugin i rozšíření byly správně zobrazeny v seznamu. Úspěšně proběhla i odinstalace.

5.2.2 Inicializace pluginu

Inicializace libpurple probíhala na všech platformách v pořádku. Jako ověření posloužil zobrazený seznam protokolů, který obsahoval všechny buildované i plugin protokoly.

5.2.3 Přihlášení ke službě

Zde se vyskytly u kolegů z ExtBrain komunity problémy. Byla proto implementována dílčí vylepšení a rozšířen výpis konkrétních chybových zpráv. Situace se zlepšila, ovšem k chybám stále v některých případech dochází. Analýzou chybových hlášení ani použitého prostředí se

zatím nepodařilo odhalit přesnou příčinu. Pomůže ovšem obnovení stránky nebo odhlášení a přihlášení. Problémy se dařilo řešit i instalací do nového uživatelského profilu prohlížeče.

V Linuxu mohou nastat při nesprávné konfiguraci a buildu podpory SSL problémy se zabezpečeným přihlášením. V takovém případě je zobrazeno odpovídající chybové hlášení. Kromě opravy buildu je možno ručně nastavit nezabezpečovaný přihlašovací server. Nicméně inicializace pluginu, načtení protokolů a uložených účtů a zahájení přihlašování fungovalo.

5.2.4 Textová komunikace

Příjem textových zpráv byl úspěšně testován na všech platformách. Zkoušena byla vždy série několika zpráv. Úspěšné bylo i zobrazení zpráv, které byly předtím odeslány offline klientovi.

Problémy byly zaznamenány v Linuxu s odesíláním zpráv, pravděpodobně byly způsobeny odlišným chováním mechanismu odesílacího threadu v Linuxu nebo dynamickým linkováním nesprávných verzí knihoven.

5.2.5 Seznam kontaktů

Bez problémové bylo získávání seznamu kontaktů jak ve formě čísla uživatele, tak jména. Bylo vyzkoušeno poslání žádosti o přidání včetně doprovodné zprávy. Ta se sice na druhé straně v klientovi Pidgin nezobrazila, nicméně bylo ověřeno že knihovní funkce zprávu jako argument dostává.

Pomocí knihovních funkcí společných pro všechny protokoly se nepodařilo naimplementovat přijetí či odmítnutí žádosti o přidání, nicméně událost i s kontaktem se zobrazí a na základě toho kontakt přidat lze.

5.3 JingleBreath plugin

5.3.1 Instalace pluginu

Instalace pluginu proběhla bez problému na všech testovaných platformách.

5.3.2 Inicializace pluginu

I zde dochází k občasným chybám při inicializaci a startu pluginu. Vzhledem ke stabilitě konzolových aplikací a knihoven se v případech libpurple i libjingle pluginu jedná pravděpodobně o specifický problém platformy FireBreath.

5.3.3 Přihlášení ke službě

Přihlášení ke službě proběhlo na všech testovacích platformách bez problému. Při testu bylo ověřeno, že v Google chatu je daný uživatel vidět a má povolená tlačítka pro audio i video hovor.

5.3.4 Seznam kontaktů

Seznam kontaktů se podařilo získat na všech testovaných platformách

5.3.5 Hlasová a video komunikace

Přenos audia se podařilo realizovat po Windows i pod Linuxem. Obousměrná audiokomunikace funguje při volání z aplikace do aplikace. Při volání na Google chat sice také fungují oba směry streamingu, ale pouze do okamžiku potvrzení hovoru. Dojde při tom ke krátkému příjmu evidentně chybných zvukových dat (zapraskání). Chyba bude pravděpodobně způsobena mírně rozdílnou implementací signálního protokolu v libjingle a na serveru Google talku. Jak bylo uvedeno v 3.7.2.3, implementace dosud není hotova a změny pořád probíhají. V této domněnce mě utvrzuje fakt, že i v diskusích na [9] jsou v různých časových obdobích diskutovány problémy jak s komunikací dvou aplikací, tak s komunikací aplikace a Google talku. I při mých testech se podařilo zprovoznit některé kanály až po update na nejnovější verzi libjingle z repozitáře.

Kapitola 6

Závěr

Poslední kapitola shrnuje dosažené cíle a vlastní přínos této práce. Rovněž jsou popsány i známé chyby a nedostatky. Na jejich základě a na základě směru vývoje projektu ExtBrain jsou navrženy možné směry pokračování této práce.

6.1 Dosažené cíle

Při tvorbě této práce jsem se podrobně seznámil s architekturou NPAPI doplňků a způsobem jejich vytváření pomocí toolkitu FireBreath. Vzhledem k současným trendům provozovat čím dál víc aplikací na platformě webového prohlížeče jsem tím podle mého názoru získal užitečné a perspektivní zkušenosti.

Při práci s libpurple jsem se seznámil s knihovnou, která je jádrem mnoha IM programů. Musel jsem se přitom vyrovnat se skutečností, že velkou část poznatků bylo třeba hledat v nedokumentovaném cizím kódu nebo na ně přicházet vlastními experimenty. Podařilo se mi vyřešit většinu problémů vyplývajících z přechodu od konzolové aplikace k pluginu. Za hlavní přínos své práce považuji provedenou analýzu a zdokumentované funkční řešení problémů s nastavením kompilace, linkování a automatického nastavení cest a proměnných systému. Tyto poznatky jsem v takto ucelené a podrobné formě na jiných zdrojích nenalezl. Dovolují přitom, aby uživatel mohl provést instalaci pluginu nejjednodušším možným způsobem. Dalším přínosem je vlastní navržené řešení architektury threadů a hlavní smyčky tak, aby bylo možno bezpečně volat funkce knihovny z uživatelského rozhraní. V neposlední řadě se mi podařilo implementovat uživatelské rozhraní, které umožňuje základní práci s IM službou. V souladu se zadáním jsem celý postup podrobně zdokumentoval.

Práce související s knihovnou libjingle byly technicky i časově nejnáročnější částí. Samotná knihovna je sice dobře zdokumentovaná, ovšem neobsahuje funkční implementaci enginu pro streaming médií z živých zařízení. Bylo potřeba sesbírat řadu poznatků z webových zdrojů i přímá komunikace s jinými vývojáři. I tak jsem musel strávit spoustu času experimentováním s nastavením a vlastními úpravami kódu. Také bylo nutné implementovat vlastní úpravy mechanismu volání knihovních funkcí z uživatelského rozhraní doplňku, které jsem navrhl. Podařilo se tedy zprovoznit obousměrné volání z aplikace na aplikaci a alespoň jednosměrný přenos videa. Navíc jsem zprovoznil testovací zasílání krátké textové zprávy. Vytvořil jsem základní uživatelské rozhraní, které umožňuje přihlášení, zahájení a ukončení hovoru. Postup je podrobně zdokumentovaný a jsou popsány pravděpodobné příčiny pro-

blémů.

6.2 Známé nedostatky

Na konkrétních počítačích kolegů z týmu ExtBrainu byly u PurpleBreath pluginu zaznamenány problémy s inicializací doplňku a přihlášením. I přes provedené úpravy, které situaci zlepšily, se problémy nepodařilo odstranit úplně.

Při specifických konfiguracích již nainstalovaných závislých knihoven jiných verzí mohou mít aplikace problém s funkčností, tomu jsem čelil přibalením požadovaných verzí do instalátoru.

Plugin PurpleBreath má v prostředí Linuxu v některých případech problémy s odesíláním zpráv, pravděpodobně jsou byly způsobeny odlišným chováním mechanismu odesílacího threadu v Linuxu nebo dynamickým linkováním nesprávných verzí knihoven.

LinphoneMediaEngine funguje ve spojení s Google chatem pouze jednosměrně, pravděpodobně kvůli odlišnému signalizačnímu protokolu. Vše je v práci podrobně popsáno.

JingleBreath plugin má pouze základní uživatelské rozhraní. Bohužel se jej z důvodu náročnosti implementace LinphoneMediaEngine nepodařilo více rozvinout. Nedostatečná je informovanost uživatele o aktuálním stavu přihlášení nebo hovoru, případně přesná chybová hlášení.

6.3 Možné pokračování práce

Tato práce je od začátku určena k tomu, že se na jejím základě bude implementovat podpora IM, audio a video komunikace do nadřazeného projektu ExtBrain Communicator. Při tom bude uzpůsoben vzhled rozhraní a mohou být přidány nové funkce. To je tedy hlavním směrem pokračování.

Zároveň se nabízí možnost provést podrobnější analýzu mechanismů nahrávání a inicializace pluginu a odladit zmíněné problémy. Jako možné cesty vidím bližší analýzu nižších vrstev FireBreath toolkitu a analýzu prostředí uživatelského profilu prohlížeče, konfliktních verzí knihoven v jiné aplikaci, softwaru bránící přístupu na určité porty apod.

Další zajímavá možnost pokračování je sledování vývoje a hlubší analýza protokolu Jingle, knihovny libjingle a její LinphoneMediaEngine a Mediastreameru. Na tomto základě potom vyřešení problémů s obousměrným přenosem audia i videa jak proti Google chatu, tak proti aplikacím.

Literatura

[1] FIREBREATH. JSAPI Properties — online manuál, 2012. http://www.firebreath.org/display/documentation/JSAPI+Properties, stav ze 10. 5. 2012.

[2] FIREBREATH. JSAPI types — online manuál, 2012. http://www.firebreath.org/display/documentation/Supported+JSAPI+types, stav ze 10. 5. 2012.

[3] GOOGLE. libjingle licence — online stránka, 2012. https://developers.google.com/talk/libjingle/license, stav ze 10. 5. 2012.

[4] GOOGLE. Google Talk XMPP extensions — online manuál, 2012. https://developers.google.com/talk/jep_extensions/extensions, stav ze 10. 5. 2012.

[5] GOOGLE. libjingle plugins — online manuál, 2012. https://developers.google.com/talk/libjingle/, stav ze 10. 5. 2012.

[6] GOOGLE. Issue 325 about call program communication — online dokumentace, 2012. http://code.google.com/p/libjingle/issues/detail?id=178, stav ze 10. 5. 2012.

[7] GOOGLE. Issue 325 about call program communication — online dokumentace, 2012. http://code.google.com/p/libjingle/issues/detail?id=325, stav ze 10. 5. 2012.

[8] GOOGLE. libjingle basic concepts — online manuál, 2012. https://developers.google.com/talk/libjingle/important_concepts, stav ze 10. 5. 2012.

[9] GOOGLE. libjingle source code — online repozitář, 2012. http://code.google.com/p/libjingle/, stav ze 10. 5. 2012.

[10] GOOGLE. MediaEngine class — online refernce, 2012. https://developers.google.com/talk/libjingle/reference/mediaengine, stav ze 10. 5. 2012.

[11] GOOGLE. NPAPI plugins — online manuál, 2012. http://code.google.com/chrome/extensions/npapi.html, stav ze 10. 5. 2012.

[12] GOOGLE. VoiceChannel class — online refernce, 2012. https://developers.google.com/talk/libjingle/reference/voicechannel, stav ze 10. 5. 2012.

[13] GOOGLE. Windows Installer XML toolset — online manuál, 2012. http://wix.sourceforge.net/, stav ze 10. 5. 2012.

[14] LINUX. ldd — Linux Programmer's Manual, 2012. http://www.kernel.org/doc/man-pages/online/pages/man1/ldd.1.html, stav ze 10. 5. 2012.

[15] LINUX. Shared Library Search Paths — online manuál, 2012. http://www.eyrie.org/~eagle/notes/rpath.html, stav ze 10. 5. 2012.

[16] MICROSOFT. Dynamic-Link Library Search Order — online manuál. <http://msdn.microsoft.com/en-us/library/windows/desktop/ms682586%28v=vs.85%29.aspx>, 2012. , stav ze 10. 5. 2012.

[17] MICROSOFT. Popis používání a chybových zpráv nástroje Regsvr32 — online manuál, 2012. http://support.microsoft.com/kb/249873/cs, stav ze 10. 5. 2012.

[18] MOZILLA. Extensions — online vývojářský manuál, 2012. https://developer.mozilla.org/en/Extensions, stav ze 10. 5. 2012.

[19] MOZILLA. Install rdf — online manuál, 2012. http://kb.mozillazine.org/Install.rdf, stav ze 10. 5. 2012.

[20] MOZILLA. Gecko plugin API reference — online manuál, 2012. https://developer.mozilla.org/en/Gecko_Plugin_API_Reference, stav ze 10. 5. 2012.

[21] MOZILLA. Plugins — online vývojářský manuál, 2012. https://developer.mozilla.org/en/Plugins, stav ze 10. 5. 2012.

[22] MOZILLA. Plugin side API reference — online manuál. <https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Plug-in_Side_Plug-in_API>, 2012. , stav ze 10. 5. 2012.

[23] MOZILLA. XPCOM — online vývojářský manuál, 2012. https://developer.mozilla.org/en/XPCOM, stav ze 10. 5. 2012.

[24] MOZILLA. XPI — online vývojářský manuál, 2012. https://developer.mozilla.org/en/XPI, stav ze 10. 5. 2012.

[25] MOZILLA. XPInstall — online vývojářský manuál, 2012. https://developer.mozilla.org/en/XPInstall, stav ze 10. 5. 2012.

[26] PIDGIN. Pidgin SSL FAQ — online manuál, 2012. http://developer.pidgin.im/wiki/FAQssl, stav ze 10. 5. 2012.

[27] PROJEKTU, P. FireBreath framework — online manuál, 2012. http://http://www.firebreath.org, stav ze 10. 5. 2012.

[28] PřISPěVATELé. Update linphone support in linjingle — online dokumentace, 2012. http://webrtc-codereview.appspot.com/404005/, stav ze 10. 5. 2012.

[29] Přispěvatelé Wikipedie. *Jingle* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://en.wikipedia.org/wiki/Jingle_(protocol)>.

[30] Přispěvatelé Wikipedie. *MIME* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://en.wikipedia.org/wiki/Internet_media_type>.

[31] Přispěvatelé Wikipedie. *Plug-in* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://en.wikipedia.org/wiki/Plug-in_(computing)>.

[32] Přispěvatelé Wikipedie. *RTCP* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://en.wikipedia.org/wiki/Real_time_control_protocol>.

[33] Přispěvatelé Wikipedie. *SRTP* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://en.wikipedia.org/wiki/Secure_Real-time_Transport_Protocol>.

[34] Přispěvatelé Wikipedie. *STUN* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://cs.wikipedia.org/wiki/STUN>.

[35] Přispěvatelé Wikipedie. *NPAPI* [online]. 2012. [cit. 12. 5. 2012]. Dostupné z: <http://cs.wikipedia.org/wiki/Netscape_Plugin_Application_Programming_Interface>.

[36] RUAN, Y. Rozšíření ExtBrain Communicatoru o další komunikační protokoly — bakalářská práce, 2012.
http://extbrain.felk.cvut.cz/theses/bp-yun-ruan.pdf, stav ze 10. 5. 2012.

[37] web:bonjour. Bonjour — online dokumentace, 2012.
http://developer.apple.com/networking/bonjour/download/, stav ze 10. 5. 2012.

[38] web:extbrain. Extbrain — online dokumentace, 2012.
http://extbrain.felk.cvut.cz/, stav ze 10. 5. 2012.

[39] web:gcwebrtc. WebRTC — online repozitář, 2012.
http://code.google.com/p/webrtc/, stav ze 10. 5. 2012.

[40] web:glibeventloop. The Main event loop — online manuál, 2011.
http://developer.gnome.org/glib/2.30/glib-The-Main-Event-Loop.html, stav ze 10. 5. 2012.

[41] web:gtkthread. GTK Thread — online manuál, 2012.
http://www.gtk.org/api/2.6/gdk/gdk-Threads.html, stav ze 10. 5. 2012.

[42] web:libpurplecom. Libpurple — online manuál, 2011.
http://www.libpurple.com/, stav ze 10. 5. 2012.

[43] web:linphone. Linphone — online manuál, 2012.
http://www.linphone.org/, stav ze 10. 5. 2012.

[44] web:mediastreamer. Mediastreamer2 — online manuál, 2012.
http://www.linphone.org/eng/documentation/dev/mediastreamer2.html, stav ze 10. 5. 2012.

[45] web:msaudiostream. Mediastreamer2 AudioStream — online manuál. <http://download-mirror.savannah.gnu.org/releases/linphone/mediastreamer/doc/group__audio__stream__api.html>, 2012. , stav ze 10. 5. 2012.

[46] web:ortp. oRTP — online manuál, 2012. http://www.linphone.org/eng/documentation/dev/ortp.html, stav ze 10. 5. 2012.

[47] web:pidgin. Pidgin — domovská stránka, 2012. http://pidgin.im, stav ze 10. 5. 2012.

[48] web:pidgindev. Pidgin vývojářská dokumentace — online manuál, 2012. http://develope.pidgin.im, stav ze 10. 5. 2012.

[49] web:pidgininstall. installing Pidgin — online manuál, 2012. http://developer.pidgin.im/wiki/Installing Pidgin, stav ze 10. 5. 2012.

[50] web:pidgininstallwin. installing Pidgin — online manuál, 2012. http://http://developer.pidgin.im/wiki/BuildingWinPidgin, stav ze 10. 5. 2012.

[51] web:pidginlibpurpledoc. Libpurple dokumentace — online manuál, 2012. http://developer.pidgin.im/doxygen/2.7.11/html/group__core.html, stav ze 10. 5. 2012.

[52] web:pidginpurple. What is Libpurple — online manuál, 2012. http://developer.pidgin.im/wiki/WhatIsLibpurple, stav ze 10. 5. 2012.

[53] web:rtp. RTP — online manuál, 2012. http://www.networksorcery.com/enp/protocol/rtp.htm, stav ze 10. 5. 2012.

[54] web:talkdevel. Google Talk for Developers — online manuál, 2012. https://developers.google.com/talk/open_communications, stav ze 10. 5. 2012.

[55] web:webrtc. WebRTC — online manuál, 2012. http://www.webrtc.org/, stav ze 10. 5. 2012.

[56] web:xmpp. XMPP — online manuál, 2012. http://xmpp.org/, stav ze 10. 5. 2012.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

GUI Graphical user interface

HTML HyperText Markup Language

ICQ Služba pro komunikaci prostřednictvím textových zpráv

IM Instant Messaging

MIME Multipurpose Internet Mail Extensions

NPAPI Netscape Plugin Application Programming Interface

SDK Software development kit

SSL Secure Sockets Layer

NAT Network address translation

NSPR Netscape Portable Runtime

NSS Network Security Services

oRTP Real-time Transport Protocol

PCM Pulse-code modulation

SIP Session Initiation Protocol

SRTP Secure Real-time Transport Protocol

RTP Real-time Transport Protocol

STUN Session Traversal Utilities for NAT

TCP Transmission Control Protocol

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

XMPP Extensible Messaging and Presence Protocol

XPCOM Cross Platform Component Object Model

XPI Cross-Platform Installer Module

XUL XML User Interface Language

⋮

Příloha B

Instalační a uživatelská příručka

Postup pro instalaci pluginu, přihlášení se ke službě a komunikaci.

B.1 PurpleBreath

Do prohlížeče nainstalujeme rozšíření pomocí instalátoru `PurpleBreathInstaller.xpi`. Zkontrolujeme v seznamu pluginů prohlížeče, že je přítomen PurpleBreath plugin.

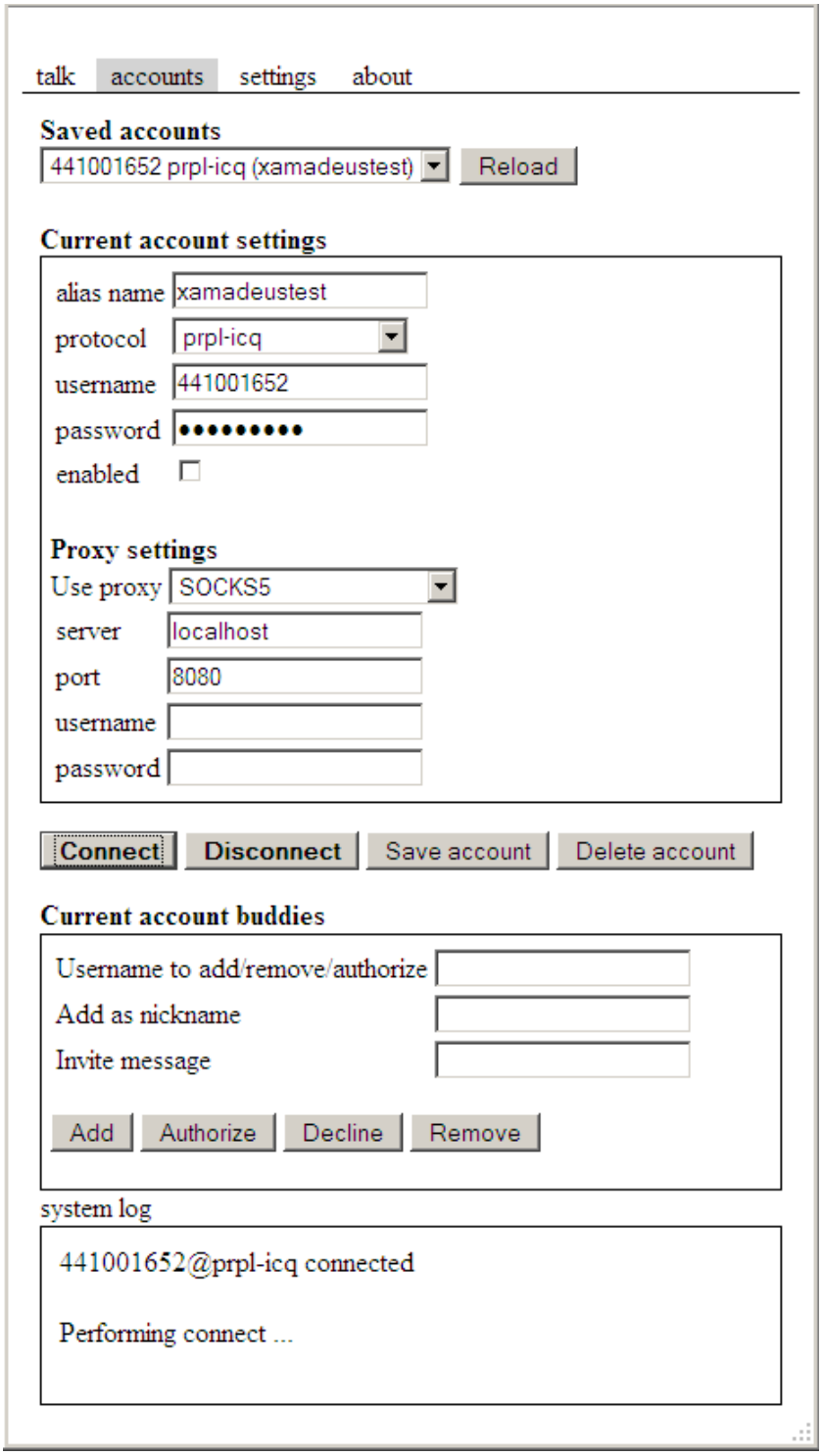
Otevřeme uživatelské rozhraní a přejdeme na oddíl `accounts`. Vytvoříme nebo upravíme uložené účty. Alias name je pouze název, pod jakým se bude účet zobrazovat. Pokud požadujeme nastavení proxy, vybereme a vyplníme příslušný typ. Úpravu každého účtu je potřeba uložit pomocí tlačítka `Save account`. Poté je možné se přihlásit tlačítkem `Connect`. Připojování již připojené aplikace zobrazí varovnou hlášku. V případě problému s přihlášením, zejména při změnách v účtech, vyzkoušíme se odpojit tlačítkem `Discnnect` a nově se připojit. Úspěšné přihlášení je zaznamenáno v okně system logu, stejně jako případná konkrétní chybová hláška. Pro přidání nebo smazání kontaktů v seznamu slouží okno Current account Buddies (obr. B.1).

Po přihlášení můžeme kounikovat v oddílu `talk`. Zvolíme ze seznamu uživatele, napíšeme zprávu a odešleme tlačítkem `Send`. Příchozí zprávy jsou zobrazovány ve spodním okénku (obr. B.2).

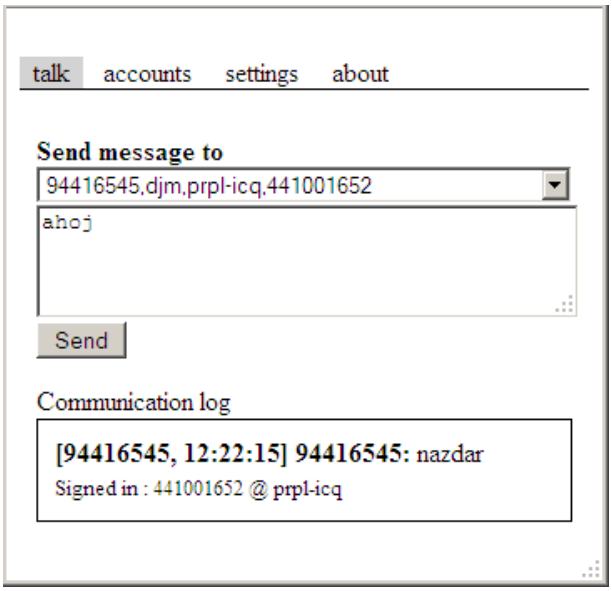
B.2 JingleBreath

Do prohlížeče nainstalujeme rozšíření pomocí instalátoru `JingleBreathInstaller.xpi`. Zkontrolujeme v seznamu pluginů prohlížeče, že je přítomen JingleBreath plugin.

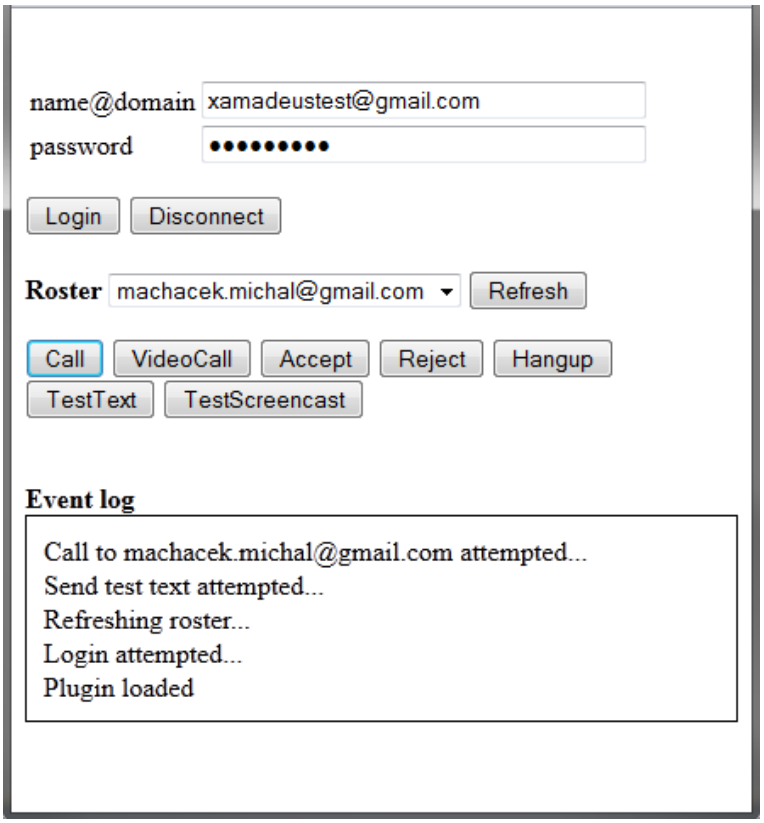
Ke službě se přihlásíme po zadání uživatelského jména a hesla. Jméno je ve tvaru `user@gmail.com`. Po přihlášení tlačítkem `Login` provedeme stiskem `Refresh` aktualizaci seznamu online kontaktů. Vybereme konkrétního uživatele a zahájíme buď audio nebo video hovor příslušnými tlačítky. Funkce accept a reject slouží k přijmutí nebo odmítnutí příchozího hovoru. Hangup zavěšuje aktuálně probíhající hovor. Lze vyzkoušet poslání krátké testovaní textové zprávy tlačítkem `testText`. Viz obr. B.3.



Obrázek B.1: Správa účtů v rozhraní PurpleBreath



Obrázek B.2: Textová komunikace pomocí rozhraní PurpleBreath



Obrázek B.3: Uživatelské rozhraní JingleBreath

Příloha C

Dokumentace PurpleBreath API

V následující tabulce jsou popsány všechny implementované funkce JavaScript API.

`Connect()`
Spustí běh MainEventLoop v knihovně libpurple, uvede účty do Online stavu. Chybu vrátí v případě že aplikace již byla připojena.

`Disconnect()`
Uvede účty do Offline stavu a přeruší běh hlavní smyčky MainEventLoop.

`Version()`
Vrací řetězec s verzí libpurple

`Send(fromproto, fromname, toname, text)`
`std::string fromproto` Název protokolu odesílatele
`std::string fromname` Uživatelské jméno odesilatele
`std::string toname` Uživatelské jméno příjemce
`std::string text` Text zprávy
Pošle z účtu specifikovaného jménem a protokolem zprávu zvolenému kontaktu

`SaveAccount(alias, proto, username, password)`
`std::string alias` User friendly jméno účtu
`std::string proto` Protokol účtu
`std::string username` Uživatelské jméno
`std::string password` Heslo
Uloží účet daných parametrů (bez informace o proxy)

`SaveAccountWProxy(alias, proto, username, password,`
`proxytype, proxyserver, proxyport, proxyuser, proxypasswd)`
`std::string alias` User friendly jméno účtu
`std::string proto` Protokol účtu
`std::string username` Uživatelské jméno
`std::string password` Heslo

`int proxytype` Čislo typu proxy
`std::string proxyserver` Adresa proxy serveru
`int proxyport` Port proxy serveru
`std::string proxyuser` Uživatelské jméno k proxy serveru
`std::string proxypasswd` Heslo k proxy serveru
Uloží účet daných parametrů s informací o proxy

`DeleteAccount(name, proto)`
`std::string name` Uživatelské jméno
`std::string proto` Protokol
Smaže daný uložený účet

`FB::VariantList GetSavedAccounts()`
Vrací seznam všech uložených účtů

`FB::VariantList GetProtocols()`
Vrací seznam všech podporovaných protokolů

`FB::VariantList GetAccountBuddies(username, proto)`
`std::string username` Uživatelské jméno
`std::string proto` Protokol
Vrací seznam všech kontaktů z daného účtu

`FB::VariantList GetAllBuddies()`
Vrací seznam všech kontaktů ze všech účtů

`FB::VariantList GetAccountDetails(username, proto)`
`std::string username` Uživatelské jméno
`std::string proto` Protokol
Vrací všechny detaily o daném uloženém účtu včetně případné proxy

`FB::VariantList GetPluginDetails()`
Vrací strukturovaný seznam s nastavením a parametry pluginu

`AddBuddy(proto, accusername, usernametoadd, nicktoaddas, message)`
`std::string proto` Protokol účtu do kterého se přidává
`std::string accusername` Uživatelské jméno účtu do kterého se přidává
`std::string usernametoadd` Uživatelské jméno které se přidává
`std::string nicktoaddas` Přezdívka pod kterou se přidává
`std::string message` Zpráva uživateli
Přidá uživatele do seznamu kontaktů a pošle přidávanému zprávu se žádostí

`RemoveBuddy(proto, accusername, usernametoadd)`
`std::string proto` Protokol účtu ze kterého se odebírá
`std::string accusername` Uživatelské jméno účtu ze kterého se odebírá
`std::string usernametoadd` Uživatelské jméno které se odebírá

Odstraní z kontaktů daného účtu specifikovaného uživatele

`AuthorizeBuddy(proto, accusername, usernametoadd)`
`std::string proto` Protokol účtu ze kterého se povoluje
`std::string accusername` Uživatelské jméno účtu ze kterého se povoluje
`std::string usernametoadd` Uživatelské jméno kterému se povoluje autorizace
Povoluje určenému uživateli přidání do kontaktů

`DeclineBuddy(proto, accusername, usernametoadd)`
`std::string proto` Protokol účtu ze kterého se zakazuje
`std::string accusername` Uživatelské jméno účtu ze kterého se zakazuje
`std::string usernametoadd` Uživatelské jméno které se zakazuje
Zakazuje určenému uživateli přidání do kontaktů

Příloha D

Dokumentace JingleBreath API

V následující tabulce jsou popsány všechny implementované funkce JavaScript API.

`Login(name, pass)`
`std::string name` Přihlašovací jméno
`std::string pass` Heslo
Přihlásí se ke službě pomocí zadaného jména a hesla. Jméno je ve tvaru `ucet@gmail.com`

`Disconnect()`
Odhlásí se od služby

`Call(name)`
`std::string name` Jméno volaného kontaktu
Zahájí audio hovor s daným kontaktem

`VCall(name)`
`std::string name` Jméno volaného kontaktu
Zahájí video hovor s daným kontaktem

`SendText(name,text)`
`std::string name` Jméno cílového kontaktu
`std::string text` Text zprávy
Pošle danému kontaktu textovou zprávu

`Screencast()`
(Experimentální) Zahájení screencastu

`Accept()`
Přijme příchozí hovor

`Reject()`
Přijme příchozí hovor

`Hangup()`
Zavěsí probíhající hovor

`FB::VariantList GetRoster()`
Vrátí seznam online kontaktů (roster)

Příloha E

Obsah přiloženého CD