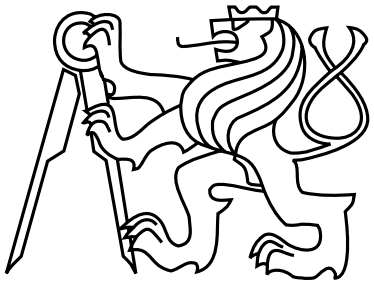
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Nástroj pro snadný návrh uživatelského rozhraní

Petr Mach

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Web a multimedia

9. ledna 2012

Poděkování

Rád bych poděkoval rodičům, přítelkyni a kamarádům za velkou podporu během psaní mé bakalářské práce. Velké díky si zaslouží také vedoucí práce Ing. Tomáš Novotný za jeho vynikající přístup, ochotu a trpělivost.

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 Praze dne 6. 1. 2012 .....

Abstract

The aim of this bachelor project is to implement a tool for designing and prototyping of a user interface, that will also allow creating of new custon elements. Source files of this tool will be synchronized with a remote storage. Part of the focus of this thesis is creating a module for connecting and communication with the remote storage.

Abstrakt

Cílem této bakalářské práce je vytvořit nástroj pro návrh a prototypování uživatelského rozhraní, který bude zároveň umožňovat vytváření šablon vlastních ovládacích prvků. Současně bude možné zdrojové soubory nástroje načítat nebo ukládat ve vzdáleném úložišti podle standardu IMAP. Součástí práce je vytvoření modulu pro připojení ke vzdálenému úložišti a komunikaci s ním.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době grafických aplikací se při návrhu softwaru klade stále větší důraz na grafickou atraktivitu rozhraní, se kterým bude uživatel pracovat. Kritickou součástí vývoje se tak stává *návrh uživatelského rozhraní*, tedy dokument znázorňující rozložení všech prvků v rámci uživatelského rozhraní programu. Ten pak slouží programátorům jako předloha pro samotnou implementaci.

Návrh uživatelského rozhraní má několik cyklů[3]. Během nich se tento návrh vylepšuje a jeho finální podoba by měla téměř odpovídat vzhledu výsledného produktu. Na samotném začátku vývoje může takovýto návrh tvořit pouze několik čtverců nakreslených tužkou na papíře, kde každý udává polohu některého funkčního celku, např. menu u webové stránky. Takto zjednodušené uživatelské rozhraní je jednoduché vytvořit přímo během konzultace s klientem, získat od něj okamžitě zpětnou vazbu a případně návrh pozměnit. Tato činnost se také nazývá *prototypování*. Během vývoje jsou poté jednotlivé funkční celky podrobněji rozkresleny a případně opět doladěny podle přání klienta. Po dostatečném množství těchto iterací je pak finální návrh předán grafikovi k vytvoření konečné podoby uživatelského rozhraní systému. Důležité je, že programátor může po každé z iterací obdržet poslední verzi návrhu a okamžitě ji zapracovat, aniž by musel čekat na finální podobu. To samozřejmě šetří čas potřebný pro vývoj.

Existuje řada způsobů návrhu uživatelského rozhraní[13]. Jednou z nejoblíbenějších metod je stále kreslení rukou na papír. Jeho hlavní předností je rychlost a intuitivita takového návrhu, což je ale výhodné pouze na samém začátku vývoje, kdy ještě není výsledek příliš komplexní. Nevýhodou kreslení rukou je poměrně komplikovaný postup, pokud je nutné měnit pozici nakreslených prvků. To se v některých případech řeší vystřihnutím papírových "komponentů"a jejich následnou manipulací, ale častěji se však používají softwarové nástroje.

Programy pro návrh uživatelského rozhraní jsou dnes, v době miniaturních notebooků, už téměř stejně rychle použitelné jako zmíněná tužka a papír. Jejich hlavní výhodou je pak jednodušší práce s výsledným návrhem, jeho mnohem lepší vzhled a snažší archivace i distribuce. Běžně dostupné nástroje však pracují se zdrojovými soubory pouze lokálně. Pokud je tedy potřeba zdrojové soubory distribuovat, možnosti jsou např. odeslat soubor pomocí elektronické pošty nebo jej nahrát do nějakého typu úložiště, ať se již jedná o přenosný disk nebo FTP server.

Jedním z nástrojů pro návrh uživatelských rozhraní je i program Pencil [2]. Zadání této bakalářské práce umožňuje pokračovat ve vývoji již existujícího řešení, a po zvážení poža-

davků bylo rozhodnuto využít právě program Pencil. Toto rozhodnutí bylo učiněno především díky vynikajícím předchozím zkušenostem s Pencilem z mé strany, i ze strany vedoucího této práce. Druhou částí zadání je vytvořit synchronizační modul pro Pencil, který odstraní výše popsané potíže s distribucí zdrojových souborů. Z tohoto důvodu byla vytvořena komponenta pro připojení k serveru pomocí protokolu `IMAP`[1], pracovně pojmenovaná `libimap`, která poslouží k dalšímu vývoji komponent projektu ExtBrain Communicator[10].

V následující kapitole 2 je popsán problém, který se tato práce snaží řešit, dále jsou vymezeny a odůvodněny cíle této práce. Ty jsou poté shrnuty do formálních požadavků a je nastíněna struktura práce.

V kapitole 3 je blíže popsána využívaná aplikace Pencil na začátku práce a požadované změny. Tato kapitola zároveň popisuje principy a specifika protokolu `IMAP` potřebná pro implementaci.

Kapitoly 4 a 5 se věnují technické realizaci této práce, tedy vytvořené knihovně `libimap`, resp. modifikacím programu Pencil. Implementační sekce této práce byla pro větší přehlednost rozdělena do dvou samostatných celků. Důvodem byl jednak rozsah potřebný k popisu obou částí, zároveň je však takto patrné, že knihovna `libimap` je úplně oddělena od aplikace Pencil.

V kapitole č. 6 jsou podrobně rozebrány kroky provedené ohledně testování a ladění aplikace. Kromě standartního uživatelského testování práce je zde popsána také testovací komponenta, vytvořená během implementace knihovny `libimap`, a která přímo její součástí.

Kapitola 7 nastiňuje některé možnosti a doporučení, jak na tuto práci navázat v dalším vývoji, a to jak co se týká programu Pencil, tak i knihovny `libimap`.

Kapitola 2

Popis problému a specifikace cíle

Hlavním výstupem práce je program pro snadný návrh uživatelského rozhraní, který by kromě běžných funkcí pro práci s grafikou měl umožňovat i jednoduchou definici vlastních prvků. Důraz je kladen hlavně na rychlost práce, flexibilitu ovládání a podpůrných funkcích pro co možná největší urychlení procesu návrhu.

Jak bylo zmíněno v 1. kapitole, běžné nástroje pro návrh uživatelských rozhraní pracují pouze s lokálními soubory a distribuce musí probíhat např. zasláním souboru elektronickou poštou atp. Proces návrhu uživatelského rozhraní by bylo možné ještě dále urychlit, pokud by mohla být distribuce (a archivace) rychlá a intuitivní přímo z nástroje.

2.1 Specifikace cíle

Jelikož zadání umožňuje použít existující řešení, nebylo nutné vytvářet jej od začátku a stačilo zvolit vhodný nástroj k rozšíření. Nástroj musí být podporován technologiemi použitými v projektu ExtBrain Communicator. To je program zaměřený na komunikaci a spolupráci uživatelů a je součástí programu Mozilla Thunderbird[8]. Z toho vyplývá použití technologií Mozilla Application Frameworku.

2.2 Požadavky

Z předchozích částí této kapitoly byly formulovány následující požadavky na aplikaci.

2.2.1 Funkční požadavky

1. Nástroj umožní návrh uživatelského rozhraní.
2. Nástroj umožní definici vlastních ovládacích prvků.
3. Bude možné načítat a ukládat zdrojové soubory do vzdáleného úložiště přes protokol IMAP.
4. Bude možné načítat a ukládat vlastní definice ovládacích prvků do vzdáleného úložiště přes protokol IMAP.

5. Nástroj umožní základní formu synchronizace zdrojových souborů se vzdáleným úložištěm.

2.2.2 Nefunkční požadavky

1. Nástroj bude postaven na technologii podporované v projektu ExtBrain Communicator.
2. Nástroj i synchronizační komponenta budou napsány s důrazem na modularitu a snadný další vývoj.
3. Práce s nástrojem nevyžaduje permanentní připojení k Internetu.

Kapitola 3

Současný stav a návrh řešení

V této kapitole je nejprve popsán proces volby použitých technologií, dále jsou rozebrány specifika jednotlivých vyvíjených komponent, na samém konci kapitoly je pak navržen postup pří implementaci.

3.1 Volba implementačního prostředí

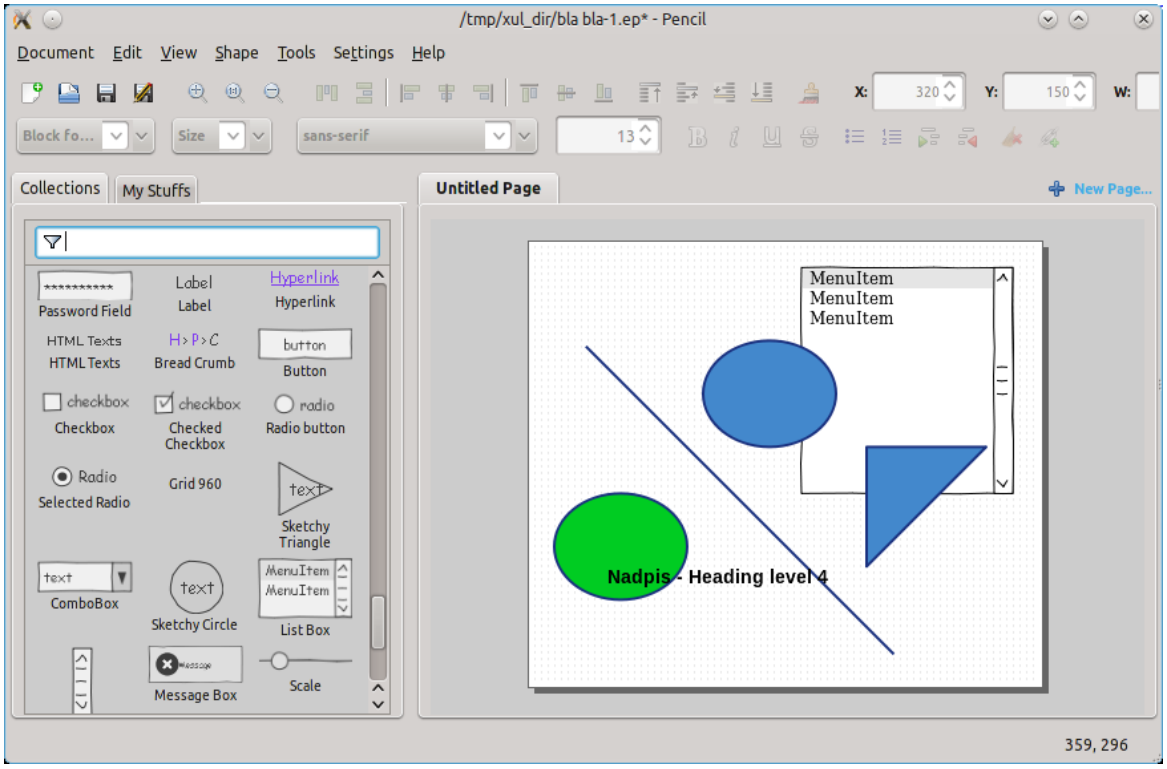
Jako implementační prostředí pro tuto práci byl určen Mozilla Application Framework (MAF), protože nástroj má být v budoucnu zařazen jako součást projektu ExtBrain Communicator. Z tohoto důvodu musí být implementován pomocí stejných technologií jako jeho ostatní součásti, tedy být napsán v jazyce JavaScript a využívat technologie Mozilla XUL. Tyto podmínky splňuje jediný existující program, a sice právě Pencil.

Co se týká synchronizačního modulu, samozřejmě musí být postaven na stejných technologiích jako nástroj pro který je určen. Jako cílový standard vzdáleného úložiště použitého pro práci s programem byl zvolen protokol internetové pošty IMAP. Důvodem je, že projekt ExtBrain již v současnosti protokol IMAP používá pro ostatní komponenty ExtBrain Communicatoru a nebude tedy potřeba zřizovat žádné speciální úložiště.

3.2 Pencil

Pencil je nástroj pro tvorbu diagramů a GUI prototypů z vývojářské firmy Evolus. Pencil je vyvíjen pod licencí GNU General Public License, má tedy otevřené zdrojové kódy. Program je postaven na Mozilla Application Frameworku a je šířen zdarma, primárně jako add-on internetového prohlížeče Mozilla Firefox, ale k dispozici je i samostatná (standalone) verze. Významnou výhodou Pencilu mezi konkurenčními programy je platformová nezávislost, jelikož MAF podporuje tři nejpoužívanější operační systémy (Windows, Linux a Mac OS) naprosto bezproblémově. S tímto programem mám zároveň vynikající zkušenosti a subjektivně ho řadím mezi nejlepší na trhu.

Na obrázku 3.1 je screenshot hlavního okna programu.



Obrázek 3.1: Snímek hlavního okna programu Pencil

3.2.1 Funkce

V následující části jsou shrnuty nejdůležitější funkce programu Pencil, které návrh uživatelského rozhraní urychlují a usnadňují.

3.2.1.1 Návrh

Program je primárně určen pro návrh uživatelských rozhraní (GUI) a vytváření jejich prototypů. K tomu jsou k dispozici předdefinované základní elementy, tzv. `stencily`. Ty jsou uspořádány do tématických kolekcí. Kolekce je dále možné vytvářet nebo importovat a během práce tak jdou využívat např. geometrické obrazce, prvky webových stránek i standartní prvky známé z rozhraní GTK nebo Windows. Samostatnou zmínku si zaslouží kolekce Sketchy GUI, která vzhledem prvků připomíná návrh rukou, a tak se pro prototypování mimořádně hodí.

Editor umožňuje všechny základní operace pro práci s vektorovou grafikou, jako je zarovnání, z-ordering (možnost umístit prvek nad, resp. pod jiný prvek), změna velikosti nebo rotace. Samozřejmostí je vytváření grafických primitiv (kruhů, obdélníků atp.), nastavení barev, přechodů nebo použití obrázků.

V rámci jednoho projektu lze pracovat na více stránkách, mezi nimiž lze nadefinovat prokliky a simulovat tak navigaci mezi jednotlivými obrazovkami při průchodu aplikací. Je

možné těmto stránkám nastavit i masky průhlednosti a dědit je mezi sebou, což návrh značně urychluje.

3.2.1.2 Export projektu

Pencil umožňuje export projektů jako obrázků, PDF dokumentů nebo HTML stránek. Zatímco první dva zmiňovaní slouží hlavně pro účely dokumentace, HTML export ve spojení s proklikem mezi okny dělá z Pencilu výkonný nástroj pro prezentaci prototypů GUI i se scénáři použití. Takto vygenerovaný prototyp lze téměř přímo použít i k uživatelskému testování návrhu GUI.

3.2.1.3 Privátní kolekce

Silnou funkcí Pencilu je i možnost jednoduše vytvářet tzv. privátní kolekce. Jakoukoliv vytvořenou skupinu prvků lze uložit jako prvek privátní kolekce a později kdykoliv použít, bez nutnosti je explicitně kopírovat. Navíc jsou takto vytvořené prvky snadno dostupné z palety.

3.3 IMAP

`Internet Message Access Protocol` (IMAP) je jedním ze dvou nejznámějších standardů pro výběr elektronické pošty. Druhým takovým standardem je `Post Office Protocol` (POP). Většina klientů pro elektronickou poštu v dnešní době podporuje oba tyto protokoly. Podpora IMAPu na straně serveru je samozřejmostí u většiny populárních poskytovatelů schránek elektronické pošty. V současnosti se používá IMAP verze 4, revize č.1, který je popsán v RFC3501[1].

Následuje stručný přehled principů a specifik protokolu IMAP, které bylo potřeba při implementaci zohlednit. Více informací a plná specifikace protokolu, viz. RFC3501[1].

3.3.1 Příkazy

Komunikace se serverem po protokolu IMAP je čistě textová a probíhá formou zaslání zprávy s klientským příkazem a obdržení zprávy se serverovou odpovědí. Každý příkaz musí být před odesláním na server ukončen správným zalomení na konci řádku, tzv. `CLRF`, tedy v JavaScriptu textovým řetězcem `\r\n`.

Na začátku všech příkazů zaslaných na server se musí nacházet tzv. *tag*, tedy identifikátor příkazu. Kromě toho, že nesmí obsahovat mezery, záleží tvar takového tagu na klientském zařízení a slouží pouze ke spárování požadavku klienta s odpovědí serveru. Nejčastěji se tagy uvádí jako alfanumerická sekvence nebo pouze v číselné formě.

Pokud je tedy uveden např. příkaz ve tvaru "PŘÍKAZ1 parametr1 parametr2" a je odeslaný jako čtvrtý v pořadí, znamená to, že na server je odesláno `03 PŘÍKAZ1 parametr1 parametr2\r\n`.

3.3.2 Odpovědi serveru

Server může poslat jednu ze tří druhů odpovědí - *status response* (informace o stavu serveru nebo provedení příkazu), *server data* (data vyžádaná od serveru) a *command continuation request* (informace o přijetí předchozí části zprávy a výzva k pokračování posílání dat). Je přesně předepsáno, jaký druh odpovědi a se použije po jakém volání. Prvně jmenovaný *status response* klient obdrží vždy.

Server může ve speciálních případech taktéž poslat zprávu nevyžádanou.

3.3.3 Stavy imapu

Po navázání spojení se klient nachází v jednom z následujících čtyř stavů:

1. Not Authenticated State
2. Authenticated State
3. Selected State
4. Logout State

Pro každý příkaz jsou definovány stavy, ve kterém jej lze použít. Příkaz může být, a většinou je, povolen ve více stavech, např. příkaz CAPABILITY lze zavolat v prvních třech, příkaz APPEND lze použít pouze v druhém a třetím.

3.3.4 Flagy

Protokol IMAP podporuje přiřazení tzv. flagů, neboli jakýchsi atributů, ke každé zprávě. Použití nejčastěji spočívá v označení přečtené, smazané nebo důležité zprávy. Některé servery zároveň používají tento mechanismus k označení zprávy s přílohou pro rychlejší manipulaci. Výhoda spočívá v možnosti tyto flagy nastavit a neměnit přitom samotné tělo zprávy.

Některé servery umožňují také vytvoření a uložení vlastních flagů. To je výhodné zejména interní potřeby klienta.

3.4 Návrh řešení

Pencil ve své standartní verzi žádnou formu synchronizace se vzdáleným úložištěm nepodporuje, je proto nutné vytvořit modul, který ji umožní.

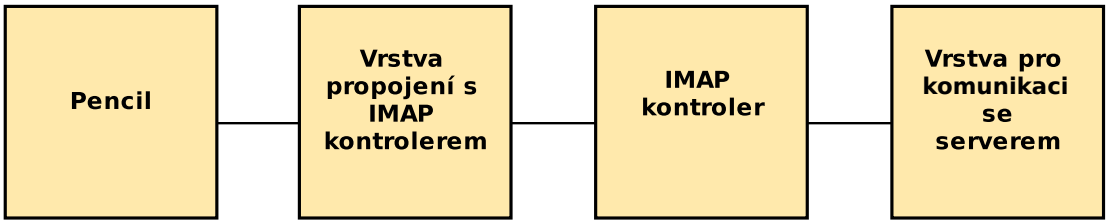
Tento musí využívat pro připojení komponentu klienta protokolu IMAP.

3.4.1 Komponenty aplikace

V následující části bude rozebrán návrh rozdělení běhu programu do komponent (vrstev) a jejich funkce. Tento model byl zvolen kvůli oddělení funkčních celků od sebe a především od programu Pencil.

1. Pencil
2. Vrstva propojení s IMAP kontrolerem
3. IMAP kontroler
4. Vrstva pro komunikaci se serverem

Tyto komponenty jsou vzájemně propojeny, komunikují spolu vždy pouze sousední vrstvy.



Obrázek 3.2: Diagram propojení komponent aplikace

3.4.1.1 Pencil

Pencil je rozšiřovaná aplikace, do které musí být na vhodná místa doplněny volání propojovací vrstvy. Prakticky se jedná o umístění ovládácích prvků na vhodné místo do grafického rozhraní aplikace a jejich přiřazení metodám propojovací vrstvy.

3.4.1.2 Vrstva propojení s IMAP kontrolerem

Tato vrstva obsahuje logiku jak Pencilu, tak IMAP kontroleru. Metoda volaná z předchozí vrstvy zde v závislosti na úkonu obstará data od aplikace, spustí kontroler a po provedení akce zpracuje výsledky. Poté se ukončí.

3.4.1.3 IMAP kontroler

Klientský kontroler pro IMAP zprostředkovává přijetí příkazů od vyšší vrstvy, jejich vykonání prostřednictvím spojení se serverem, které je popsáno níže, a navrácení výsledků.

V této vrstvě jsou definovány jednotlivé dílčí úkoly pro každou z podporovaných operací a popsány jako sekvence příkazů dle protokolu IMAP. Zároveň je nutné obstarávat parsování výsledků, aby bylo možné zjistit současný stav a určit další akci potřebnou k vykonání operace.

3.4.1.4 Vrstva pro komunikaci kontroleru se serverem

Nejnižší vrstva aplikace obsahují zprostředkovává samotnou síťovou komunikaci se serverem. Jejím učelem je obalovat serverové příkazy protokolu a v určitém rozsahu také zpracovávat přijaté výsledky.

Tato vrstva musí být co nejobecnější a naprosto nezávislá na programu Pencil, aby mohla být použita i pro jiné aplikace.

3.4.2 Princip

Cílem je využít co nejvíce z nativní funkcionality Pencilu, aby nedocházelo ke zbytečné duplikaci kódu, a jelikož Pencil již obsahuje vyzkoušené mechanismy pro ukládání a načítaní souborů lokálně, bylo by dobré je využít, namísto implementace vlastních.

Synchronizace souborů na server byla proto vyřešena následující sekvencí úkonů:

1. Převedení zdrojového souboru současného projektu na binární data
2. Upload souboru jako přílohy nové zprávy
3. Archivování předchozí verze souboru

Podobný princip existuje i pro stahování souborů ze serveru. Pouze ukládání souboru na disk musí být implementováno samostatně, protože Pencil binární data nepodporuje.

1. Výběr souboru ze souborů uložených na serveru
2. Stažení vybraného souboru
3. Uložení souboru na lokální disk
4. Otevření souboru z disku nativní metodou Pencilu

Tyto dva mechanismy fungují podobně jak pro projekty návrhu uživatelského rozhraní, tak i pro soubory s privátními kolekcemi.

Kapitola 4

libimap

Cílem práce práce bylo vytvořit implementaci klientské části protokolu `IMAP` v jazyce JavaScript, využívající komponent Mozilla Application Frameworku a tím pádem použitelný v programu Pencil a komponentách projektu ExtBrain Communicator. Při vývoji byl proto kladen velký důraz na modularitu kódu a tento klient byl implementován jako samostatná knihovna, nikoliv jako součást programu Pencil.

Knihovna byla pracovně pojmenována `libimap`.

4.1 Struktura

V této sekci je popsána struktura souborů tvořící knihovnu `libimap` a kontroler představující jednu mezivrstvu mezi knihovnou a aplikační logikou programu Pencil, viz. 3.4.1.

4.1.1 Core

Jádro knihovny libimap tvoří čtyři JavaScriptové třídy v adresáři *Core*.

```
Core/
  IMAPClient.js
  IMAPResponse.js
  MozExtbrainToolkit.js
  Util.js
```

Třída `IMAPClient` implementuje celou funkcionalitu klienta pro protokol IMAP, třída pro připojení a komunikaci po síti je zprostředkována modulem `MozExtbrainToolkit`. Třída `IMAPResponse` vytváří sktrukturu pro přijatou odpověď a stará se o základní parsování, viz 4.4. Soubor `Util` pak obsahuje pomocné funkce pro knihovnu.

4.1.2 Controller

V adresáři *Controller* se nacházejí soubory přímo využívající knihovnu `libimap`. Jedná se o mezičlánek mezi nativní funkcionalitou Pencilu a jádrem knihovny. Kromě Javascriptového kódu již tento adresář obsahuje i soubory XUL.

```
Controller/
  ImapClientController.js
  ImapConfig.js
  ImapLoadFile.js
  ImapLoadFile.xul
  IMAPParser.js
  ImapSaveFile.js
  ImapSaveFile.xul
```

4.1.3 Widget

Během vytváření Controlleru a vzorové libimap aplikace (viz 4.5) bylo vytvořeno také několik miniaturních pomocných komponent, tzv. widgetů v jazyce XUL. Ty jsou k dispozici pro použití v kontroleru k vykonání a vykonávají akce typu zadání přihlašovacích údajů k účtu nebo zobrazení seznamu zpráv. Tyto drobné komponenty jsou součástí knihovny libimap, jelikož je pravděpodobné, že ostatní aplikace využívající knihovnu je mohou potřebovat také.

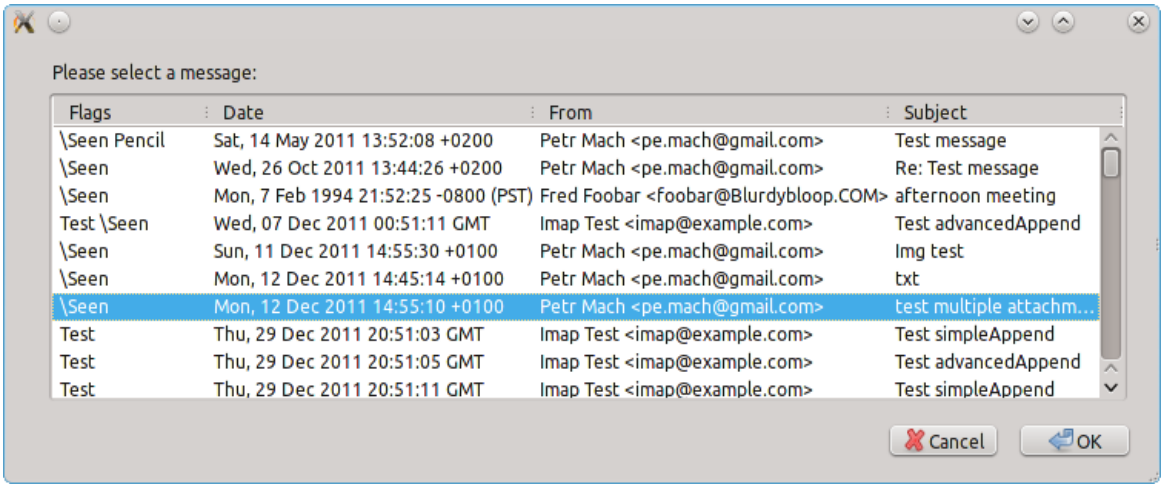
Adresář *Widget* obsahuje zdrojové soubory těchto komponent.

```
Widget/
  FolderListDialog.js
  FolderListDialog.xul
  FolderListWidget.js
  MessageListDialog.js
  MessageListDialog.xul
  MessageListWidget.js
  MessageViewDialog.js
  MessageViewDialog.xul
  MessageViewWidget.js
  ServerSpecsDialog.js
  ServerSpecsDialog.xul
  ServerSpecsWidget.js
  UserCredentialsDialog.js
  UserCredentialsDialog.xul
  UserCredentialsWidget.js
```

Příklad widgetu, konkrétně okna se seznamem zpráv ve složce, je na obrázku 4.1.

4.2 Princip

V této sekci jsou popsány principy běhu knihovny `libimap` rozdělené podle funkčních celků. Na obrázku 4.2 je struktura jádra knihovny a propojení jednotlivých tříd zachycena jako UML diagram.



Obrázek 4.1: Snímek widgetu - seznam složek

4.2.1 Připojení a komunikace

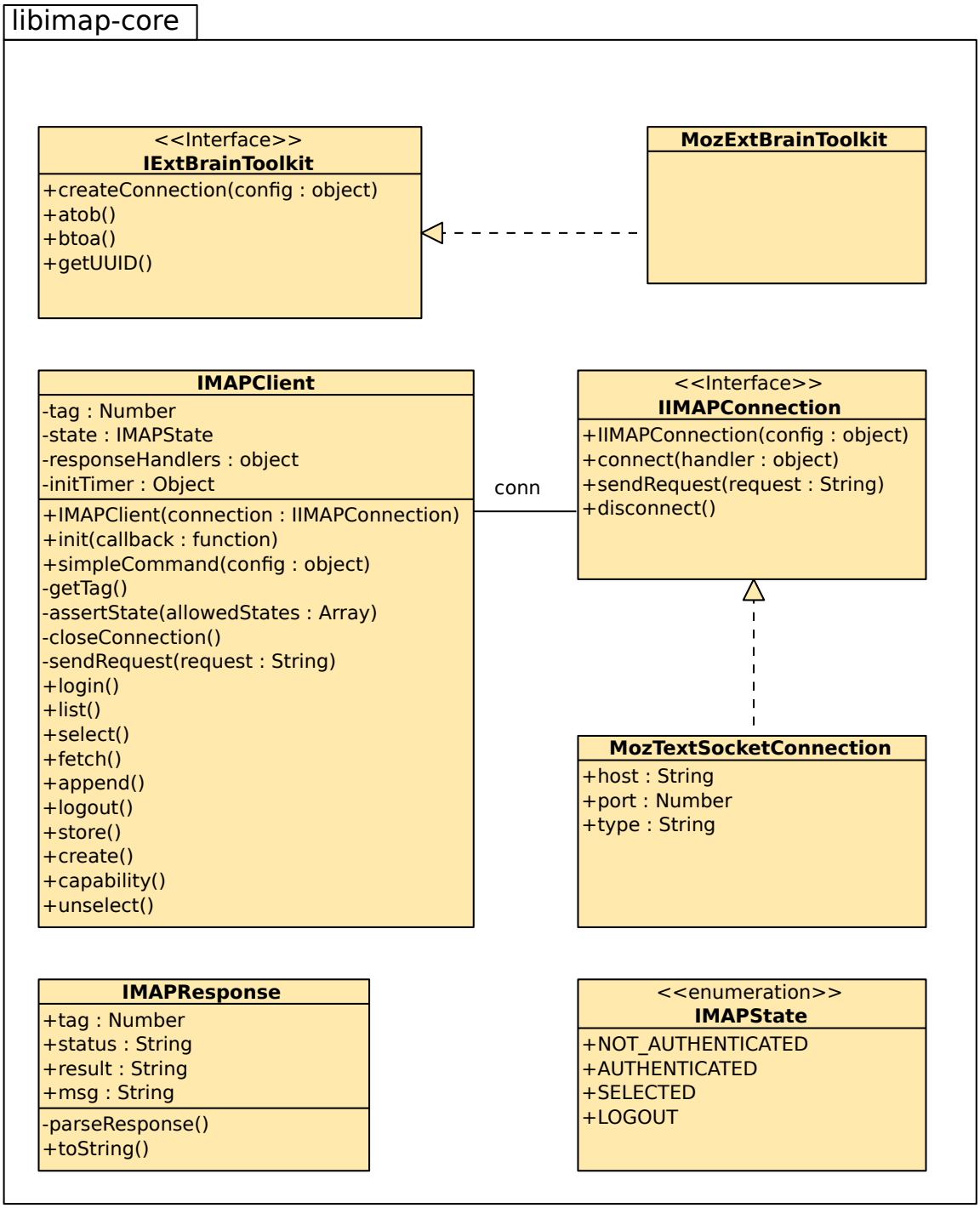
O zajištění připojení a zprostředkování komunikace se serverem v průběhu práce s klientem se stará třída vytvořená návrhovým vzorem *factory* v modulu `MozExtbrainToolkit`. Tento modul musí implementovat rozhraní `IExtbrainToolkit` a zprostředkovává funkce, které klient používá nad rámec standardního JavaScriptu. Právě kvůli zachování modularity navíc také vytváří třídu pro komunikaci po síti. Využití jiné třídy je tedy možné bez jakéhokoliv zásahu do klienta, stačí pouze v kontroleru aplikace použít jinou realizaci `IExtbrainToolkit`.

Síťová třída musí splňovat kritéria předepsaná rozhraním `IIMAPConnection`, aby s ní klient dokázal pracovat. V případě této práce je tou třídou `MozTextSocketConnection`. Ta pro svůj běh využívá `XPCOM`[9] komponentu zabývající se komunikací po síti, uváděná pod názvem *Necko*. Samotná komunikace se serverem probíhá pomocí služby `nsISocketTransportService`, která umožňuje vytvořit tzv. *transport*. Ten se připojí k serveru s nastaveným typem připojení a otevře vstupní a výstupní streamy mezi aplikací a serverem. Do výstupního streamu aplikace posílá příkazy serveru a naslouchá na vstupním streamu, kam server zasílá odpovědi.

V prvních verzích této práce byla komunikace zprostředkována synchronně, tzn. klient odeslal na server příkaz a teprve poté začal na vstupním streamu čekat na odpověď. Tento přístup se hned na začátku testování ukázal jako naivní a nedostačující, viz 6.2. Proto musela být celá připojovací třída přepsána a nyní používá asynchronní spojení mezi klientskou a serverovou částí. Toto asynchronní spojení využívá komponenty `nsIInputStreamPump` a její metody `asyncRead`. Příchozí data jsou shromažďována, dokud není odpověď kompletní, teprve pak se pomocí callbacku předá klientovi.

4.2.2 Systém volání

Kvůli výše zmíněnému asynchronnímu připojení se nedá mezi jednotlivými třídami (a často ani v jejich rámci) komunikovat standardním způsobem, kdy je kód přehledně pod



Obrázek 4.2: UML class diagram knihovny libimap

sebou v blocích a příkaz nebo volání se provede právě po skončení volání předchozího. Naproti tomu s asynchronními voláními se průběh programu kontroluje pomocí volání speciálních funkcí, tzv. callbacků. Tento přístup bohužel v JavaScriptu poměrně ztěžuje orientaci v kódu, pro korektní běh knihovny je však nezbytný.

4.2.2.1 Princip callbacku

Tento systém je podobný jako např. u technologie `AJAX`, která je v dnešní době často používaná při vývoji webových aplikací. Princip je vysvětlen na následujícím útržku kódu.

```
var callback = function(data){
    print('callback: ' + data);
    //další akce běhu programu
}

var data = getAsynchronousData(callback);
print('standardní return: ' + data);
```

Mějme metodu `getAsynchronousData`, která provádí nějakou asynchronní činnost. Touto činností může být např. čtení ze souboru, síťová komunikace atp. Metoda je navržena tak, že po skončení akce zavolá metodu callback, která jí přišla jako parametr a předá jí získaná data. Výpis programu bude vypadat následovně:

```
standardní return: undefined
callback: získaná data
```

To je způsobeno právě asynchronním přístupem metody `getAsynchronousData`. Ve chvíli, kdy má být ukončena standardním způsobem, nejsou ještě data k dispozici. Jakmile k dispozici budou, zavolá se však metoda callback a celé workflow se posune dál.

4.2.2.2 Použití callbacků v knihovně libimap

Pro předávání dat v rámci knihovny a kontroleru bylo nutné implementovat celkem tři úrovně callbacků. Na nižší úrovni je to vrstva pro propagaci přijaté serverové zprávy ze třídy `MozTextSocketConnection` do třídy `IMAPClient`. Tento callback je součástí klienta, navíc není příliš komplexní a není proto nutné jej příliš rozebírat.

Druhá (vyšší) úroveň slouží k propagaci dat mezi kontrolerem a klientem `IMAPClient` a každý kontroler, který chce klienta používat, musí tento callback implementovat v zadaném formátu. Tento callback je JavaScriptový objekt, který přijímá odpovědi od klienta a musí obsahovat metody pro následující příkazy:

- okay
- fail
- unexpected

- error

Metoda pro každý z těchto případů dostává jeden parametr. V případě prvních třech je to objekt s odpovědí, v posledním z nich pak objekt chyby, která nastala. Tento callback objekt se předává instanci třídy `IMAPClient` jako parametr v její metodě `init()`. Klient pak po přijetí každé zprávy od serveru vyhodnotí, jaká metoda callbacku se má použít a předá jí tuto zprávu rozparsovanou do struktury `IMAPResponse`. Roli ve výběru patřičné metody hraje status serverové zprávy a její tag. V implementaci kontroleru jak pro Pencil, tak i pro vzorovou aplikaci je princip takový, že metody *okay* a *fail* dál propagují přijaté odpovědi do příslušných metod callbacků třetí úrovně.

Třetí vrstva callbacku není pro funkci knihovny nutná, ale jak již bylo uvedeno, využívají ji obě implementace kontroleru v této práci. Tato vrstva se používá pro definování posloupnosti příkazů v rámci kontroleru, a tedy kontrolu toku programu. Callback je opět objekt s metodami definovanými pro stavy *okay* a *fail*. Důvody pro toto chování jsou podrobněji rozebrány v části 4.2.4.

4.2.3 Příkazy klienta

Všechny příkazy, které klient posílá na server mají ze specifikace podobnou strukturu. V rámci každého volání se pak musí vykonat následující:

- Ověření platnosti příkazu v současném stavu
- Vytvoření nového tagu
- Registrace příkazu pro pozdější manipulaci s odpovědí
- Formátování příkazu pro IMAP server
- Odeslání na server

Pro zamezení duplikování kódu, a s tím spojenou možnost zanesení chyb do programu, byla ve třídě `IMAPClient` vytvořena metoda `simpleCommand`[1]. Ta jako argument přijme konfigurační objekt klientského příkazu, a poté zformuje a navrátí metodu zajišťující všechny výše uvedené kroky.

Definice nových příkazů pro klienta je proto velmi jednoduchá, je potřeba pouze nadefinovat části specifické pro konkrétní příkaz, bez nutnosti starat se o části společné všem příkazům podle specifikace. Následující kód ilustruje použití metody `simpleCommand` na příkladu klientského příkazu `LOGOUT`

```
this.logout = this.simpleCommand({
    allowedStates : [ IMAPState.NOT_AUTHENTICATED, IMAPState.AUTHENTICATED,
        IMAPState.SELECTED ],
    command : function() {
```

[1] Název `simpleCommand` implikuje existenci metody `advancedCommand`, pro potřebu příkazů v současné verzi knihovny však implementace žádných pokročilejších mechanismů nebyla nutná. Pojmenování bylo zachováno pro případ, že v rámci budoucího vývoje by bylo potřeba takovou metodu skutečně použít.

```
            return "LOGOUT";
        },
        handler : {
            okay : function(scope, response) {
                ...
            },
            fail : function(scope, response) {
                ...
            }
        }
    });
```

4.2.4 Příkazy kontroleru

Je zřejmé, že kontroler musí definovat metody pro práci s klientem a že tato metoda nemusí obsahovat vždy jen jeden příkaz. Protože ale kontroler musí pracovat s asynchronními výstupy z klienta, bylo nutné navrhnout a implementovat speciální přístup ke kontrole toku programu. Zařazeny byly dvě metody, první pro vytvoření funkčních celků z atomických příkazu pro klienta, druhá pro vytvoření sekvence těchto celků za účelem provedení určitého úkonu.

4.2.4.1 Zřetězení příkazů

Tato metoda využívá se objektu, jež všechny dílčí kroky zapouzdřuje s tím, že tyto jednotlivé části jsou za sebe napojeny takovým způsobem, že metoda handleru *okay* jednoho bloku volá metodu *command* druhého atd. Pro jednoduchost a konzistenci je jako vstupní bod do tohoto bloku metoda o stejném názvu jako zapouzdřující objekt s prefixem „run“.

Zde je zjednodušený příklad metody „action“ kontroleru, vytvořené zřetězením nekolika příkazů pro klienta.

```
Controller.runAction = function() {
    this.action.step1.command(this);
};
Controller.action = {
    step1 : {
        command : function(scope) {
            scope.client.foo(bar);
        },
        okay : function(scope, response) {
            scope.action.step2.command(scope)
        }
    },
    step2 : {
        command : function(scope, sequenceNr) {
            scope.client.foo2(bar2);
```

```
        },
        okay : function(scope, response) { ... }
    }
};
```

Tento postup je víceméně asynchronním ekvivalentem standardního:

```
scope.client.foo(bar);
scope.client.foo2(bar2);
...
```

4.2.4.2 Fronta

V kontroleru použitém pro Pencil je implementován ještě jeden mechanismus pro definování sekvencí, a sice fronta metod k vykonání tzv. scénáře. Ta se naplní při vytváření kontroleru referencemi na jeho asynchronní bloky v požadovaném pořadí. Do kontroleru je zavedena nová metoda `runNext`, kterou vždy poslední článek každého bloku zavolá a předá tak řízení dále. Metoda `runNext` vždy vyzvedne a zavolá první metodu z fronty, jež je na řadě. Více o použití fronty viz 5.3

Následuje příklad fronty asynchronních metod kontroleru, definující scénář pro stažení seznamu zpráv:

```
c.commandQueue = [ c.runConnect, c.runSelectFolder, c.runGetMessageList,
                   c.runDisconnect ];
```

4.3 Implementované příkazy

Pro potřeby této práce nebylo nutné implementovat všechny příkazy klientské části protokolu IMAP tak, jak je definuje RFC3501[1]. Následuje seznam implementovaných příkazů v abecedním pořadí, stav, ve kterém je lze volat, jejich funkce, tvar a případně i odpovědi na ně:

4.3.1 APPEND

Příkaz `APPEND` přidá do dané složky poslaný text jako novou zprávu. Podle specifikace by text měl být ve formátu stanoveném v RFC2822[12].

Příkaz lze odeslat ve stavech `Authenticated` a `Selected`, má následující tvar. Hranaté závorky udávají nepovinné údaje.

```
APPEND <název složky> [seznam flagů] [datum/čas] <délka zprávy>
```

Server na správně formulovaný příkaz odpoví žádostí o pokračování requestu, načež klient zašle samotnou zprávu o dané délce. V případě úspěšného přidání zprávy pak server zašle zprávu `OK APPEND completed`. Více viz B.1

4.3.2 CREATE

Příkaz `CREATE` vytvoří ve schránce novou složku se zadaným jménem.

Příkaz lze volat ve stavech `Authenticated` a `Selected` v tomto tvaru:

```
CREATE <název složky>
```

4.3.3 FETCH

Volání `FETCH` vyzvedne z aktuálně vybrané složky dotázaná data.

Tento příkaz je použitelný pouze ve stavu `Selected` v následujícím tvaru:

```
FETCH <sekvence> <položky ke stažení nebo makro>
```

Parametr `sekvence` určuje, které zprávy se mají stahovat. Tvar tohoto patrametru pro všechny zprávy je `1:*`, lze používat intervaly, např. `3:7` a nebo si vyžádat pouze jednu konkrétní zprávu zadáním jejího pořadí ve složce, např. `8`.

Druhý parametr určuje formát dat, která má klient očekávat. Je možné explicitně vyjmenovat co je potřeba stahovat a není tak potřeba si vždy vyžádat kompletní data zprávy, pokud nejsou potřeba. Lze také použít definované makro, na výběr jsou tři, a sice `ALL`, `FAST` a `FULL`.

V této práci jsou použity vlastní definice položek s daty užitečnými pro použití v Pencilu a ukázkové aplikaci knihovny `libimap`. Uvedeny jsou v náledujícím bloku, první z nich se dotazuje pouze na data z hlavičky a je tak vhodný pro stahování seznamu zpráv ve složce. Druhá pak stahuje rovnou i tělo zprávy včetně příloh, hodí se proto hlavně k použití pro jedinou konkrétní zprávu a neměla být použita pro stahování intervalů zpráv nebo dokonce všech zpráv ve složce.

Dodržení těchto mechanismů optimalizuje datové přenosy mezi klientem a serverem a zajišťuje tak i ochranu běhu programu před zdánlivým *zamrznutím*, způsobeným nečinností klienta během stahování a zpracování dat[6].

Za zmínku stojí použití `BODY.PEEK` v prvním příkladě. Narozdíl od samostatného `BODY` totiž umožňuje stáhnout data zprávy, aniž by se tato zpráva automaticky označila jako přečtená.

```
(FLAGS BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT MESSAGE-ID TO)])

(FLAGS BODY[HEADER.FIELDS (DATE FROM SUBJECT MESSAGE-ID TO)] BODY[TEXT])
```

Server na příkaz odpovídá nejprve netagovanou odpovědí s vyžádanými daty a poté zakončí přenos odpovědí s tagem a statusem.

4.3.4 LIST

Pomocí příkazu `LIST` lze zobrazit seznam složek na serveru.

Příkaz je použitelný ve stavech `Authenticated` a `Selected` v následujícím tvaru:

```
LIST <reference> <název mailboxu>
```

Server na příkaz odpoví nejprve netagovaným seznamem složek vyhovujícím hledaným parametrům a poté i tagovaným oznámením o úspěšném průběhu příkazu. Pro více informací a příklady viz B.2

4.3.5 LOGIN

Protokol IMAP popisuje několik možností autentizace uživatele. Jelikož však většina IMAP serverů podporuje zašifrovaný přístup přes SSL, zabezpečení přenosu bylo přenecháno komponentě frameworku zajišťující spojení a implementovat postačilo pouze příkaz `LOGIN`. Ten posílá serveru uživatelské jméno a heslo uživatele, což by bylo nebezpečné u nešifrovaného připojení, ale je v pořádku, pokud celé spojení probíhá po SSL kanále.

Příkaz je serveru poslán ve stavu `Not Authenticated` a má tvar `LOGIN username password`

Na příkaz server odpovídá výpisem schopností serveru (`CAPABILITY`) v případě úspěchu, nebo chybovou hláškou `AUTHENTICATIONFAILED`, pokud byly zadány chybné údaje. Po úspěšném provedení tohoto příkazu se klient nachází ve stavu `Authenticated`.

4.3.6 LOGOUT

Příkaz `LOGOUT` informuje server o tom, že klient je připraven ukončit spojení se serverem. Po úspěšném provedení příkazu se klient nachází ve stavu `Logout`.

Příkaz lze odeslat ve všech stavech autentizace a má tvar `LOGOUT` bez parametrů.

Server na příkaz reaguje netagovanou odpovědí `BYE`, po které ukončuje komunikaci. Klient by měl vždy zavřít komunikační kanál teprve po obdržení této odpovědi.

4.3.7 SELECT

Příkaz `SELECT` vybere složku se zadaným jménem. Po úspěšném průběhu tohoto příkazu se klient nachází ve stavu `Selected`.

Příkaz lze volat ve stavu `Authenticated` v tomto tvaru:

```
SELECT <název složky>
```

Pokud během vybírání složky nedošlo k chybě, server odpoví sérií netagovaných dat, obsahující stav vybrané složky jako např. flagy v ní použité, počet zpráv celkem, počet nepřečtených, atp. Teprve po odeslání všech těchto informací pošle server tagovanou zprávu o úspěšném průběhu příkazu.

4.3.8 STORE

Příkaz `STORE` slouží k modifikaci dat zprávy.

Příkaz je použitelný ve stavu `Selected` v tomto tvaru:

```
STORE <sekvence> <editovaná položka> <nová hodnota>
```

Parametr `sekvence` je totožný se stejným parametrem u příkazu `FETCH`. Příkaz umí dle RFC3501 měnit pouze jedinou položku zprávy, a sice její flagy. Podporované jsou tři módy, `FLAGS`, `+FLAGS` a `-FLAGS`. První z nich nahradí zprávám jejich flagy novými hodnotami, další dva tyto nové hodnoty do stávajícího seznamu flagů přidají, resp. je odeberou.

Po úspěšném přiřazení flagů pošle server stejnou netagovanou odpověď jako u příkazu `FETCH`. Teprve poté oznámí úspěch volání.

4.4 Parsování

Jelikož veškerá komunikace probíhá textově, je nutné zprávy přícházející od serveru parsovat do použitelné podoby. Parsování má v jádře knihovny `libimap` celkem dva stupně, třetí stupeň je pak nutné zavést v rámci kontroleru klienta.

První z těchto stupňů parsování probíhá okamžitě po obdržení odpovědi od serveru a má za úkol vytvořit z textu odpovědi strukturu `ImapResponse`, se kterou je pak aplikace schopna dále pracovat. Toto počáteční parsování je přímo součástí třídy `ImapClient` a projde jím každá zachycená odpověď. První stupeň parsování musí být obecný a stejný pro všechny odpovědi, parsuje se tedy pouze status zprávy, viz 3.3.2 *status response* a zbytek je ponechán ke zpracování na později. Rozparsovaná struktura odpovědi obsahuje následující pole:

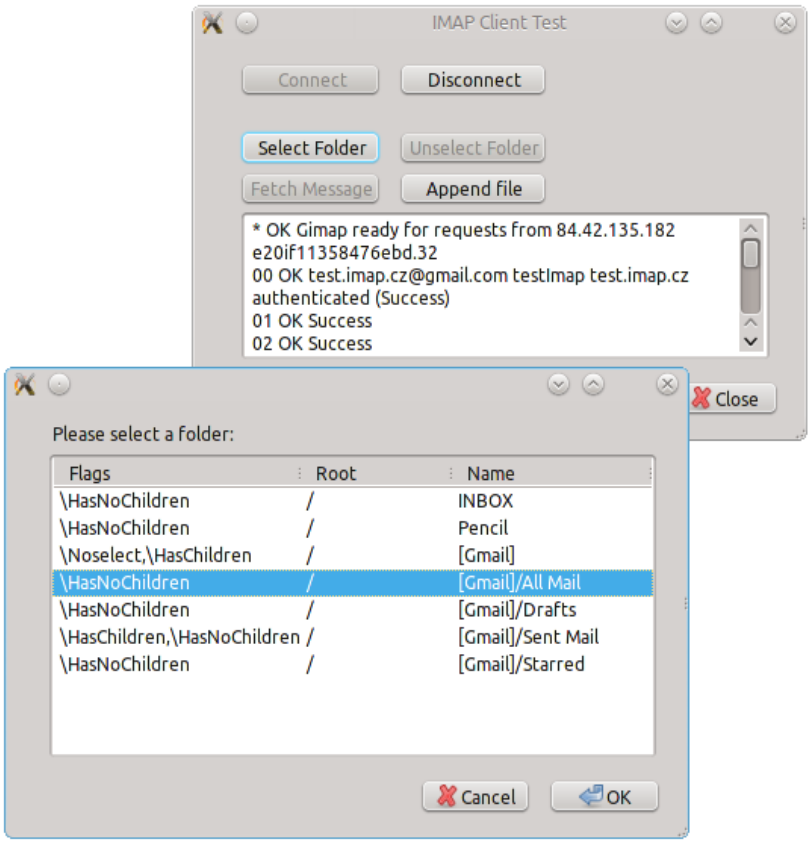
- tag
- zkratka statusu (OK, NO, BAD)
- podrobnější popis výsledku příkazu
- zbytek zprávy

Druhý stupeň parsování využívají pouze příkazy `LIST` a `FETCH`, které jako data odpovědi obdrží seznam několika položek. Tento stupeň v obou případech rozdělí tento textový seznam na jednotlivé položky před předáním rozparsované odpovědi kontroleru, u příkazu `LIST` pak dokonce rozparsuje odpověď celou. Využívá při tom faktu, že přijaté odpovědi mají přesně předepsaný formát. Z tohoto důvodu může být stejně jako první stupeň přímo součástí třídy `ImapClient`. Cílem takovéhoto designu je provádět co možná největší množství práce v jádře knihovny, společném pro všechny aplikace, které tuto knihovnu budou používat, a ulehčit tak co nejvíce tvorbu aplikačně závislého kontroleru.

Třetí (a poslední) stupeň parsování zprávy se volá přímo z kontroleru, v případě Pencilu tedy třídy `ImapClientController`. Tento stupeň využívá pouze příkaz `FETCH` a nemůže už být součástí jádra knihovny, protože vyžaduje znalost přesného formátu přijímané zprávy. Pro potřeby této práce byla vytvořena funkcionalita na parsování `IMAPParser`, která umí parsovat odpovědi na `FETCH` pro formát, v jakém jej kontroler používá.

4.5 Ukázková aplikace pro libimap

Jak již bylo uvedeno, k ovládání `libimap` knihovny je zapotřebí implementovat tzv. kontroler. Jelikož má být knihovna uvolněna k dalšímu použití, musí být kontroler součástí balíčku její distribuce. Jelikož je Pencil poměrně rozsáhlou aplikací, není vhodné, aby byl do tohoto balíčku zahrnut právě kontroler použitý pro Pencil, a bylo proto potřeba implementovat nový, ukázkový kontroler, na kterém budou demonstrovány všechny principy práce s knihovnou `libimap`.



Obrázek 4.3: Snímek okna ukázkové aplikace

Byla proto navržena a implementována jednoduchá aplikace umožňující následující scénář:

1. Připojení k serveru
2. Autentifikace serveru
3. Zobrazení seznamu složek
4. Volba konkrétní složky
5. Zobrazení seznamu zpráv ve složce

6. Stažení a zobrazení konkrétní zprávy
7. Uložení příloh zprávy
8. Přidání flagu zprávě
9. Odebrání flagu zprávy
10. Odhlášení

V této aplikaci jsou některé metody kontroleru spouštěny stiskem tlačítka, ostatní metody jsou s nimi pak spojeny do sekvence, viz 4.2.4.1. Aplikace používá všechny dostupné widgety, snaží se však z důvodu přehlednosti zůstat co nejjednodušší.

Na obrázku 4.3 je snímek okna ukázkové aplikace s použitým widgetem pro volbu složky.

4.6 libimap-console

Příkladem miniaplikace využívající knihovnu `libimap` mimo kontext Pencilu je program s pracovním názvem `libimap-console`. Jedná se o lightweight emulátor terminálu připojeného k serveru přes protokol IMAP.

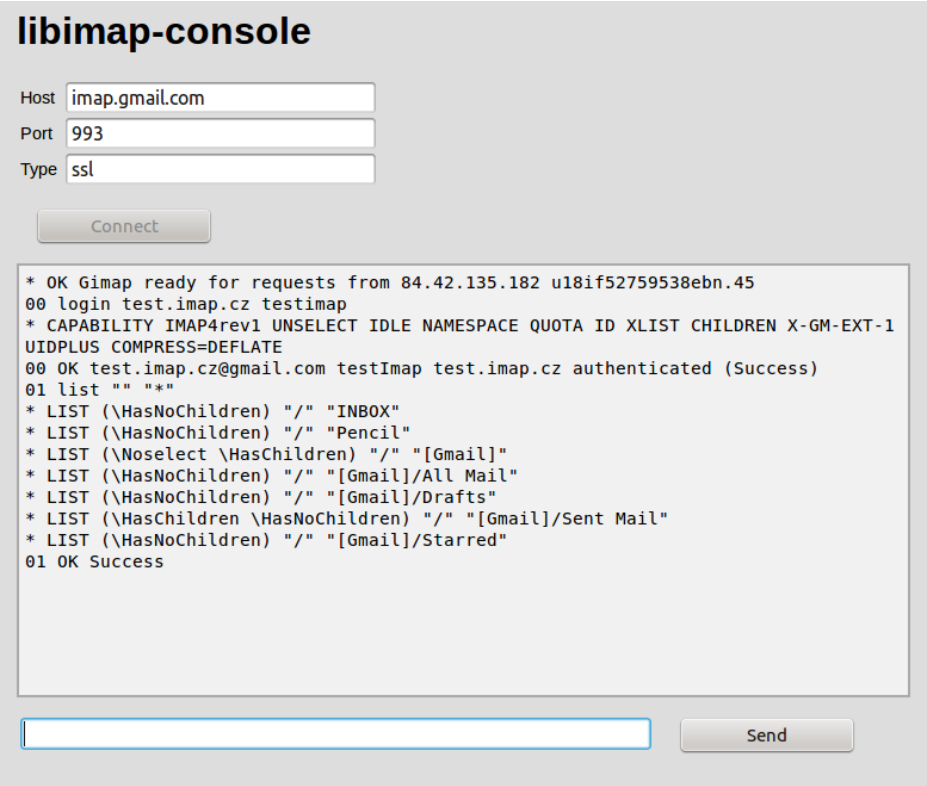
Impulsem pro vznik této aplikace byla potřeba ručně testovat připojení k serveru přes protokol IMAP a vidět tak přesně všechny klientské příkazy i odpovědi serveru. Dříve k tomuto účelu dobře sloužila aplikace PuTTY[14], ta však není nadále použitelná, protože nepodporuje zabezpečené připojení k IMAPu na portu 993 a většina IMAP serverů už přestala používat nezabezpečený port 443. Druhá alternativa, unixový open-source ssl toolkit `openssl`[11] zase v operačním systému Kubuntu Linux (jinde testován nebyl) přestává komunikovat po přijetí odpovědi na první příkaz.

Aplikace byla pojata jako HTML stránka nastylovaná pomocí CSS, s jednoduchým JavaScriptovým klientem. Jedná se o silně odlehčenou verzi knihovny libimap, která zajišťuje pouze následující funkcionalitu:

1. Navazuje spojení se serverem
2. Přijímá od uživatele textový vstup
3. Odesílá tento vstup na server
4. Tiskne serverovou odpověď do výstupního okna

Kvůli závislosti na komponentách `Mozilla XPCOM` je aplikace použitelná pouze v prohlížeči Firefox, i to je však malá cena za funkční nástroj usnadňující vývoj.

Na obrázku 4.4 je screenshot okna aplikace `libimap-console` připojené k serveru.



Obrázek 4.4: Snímek okna aplikace libimap-console

Kapitola 5

Pencil

5.1 Úpravy editoru

Ačkoliv je program Pencil již v základu velice silným nástrojem pro tvorbu GUI, samozřejmě v něm existují některé chyby a nedostatky. Část této práce se proto zabývá jejich odstraněním nebo vylepšením. Následuje seznam provedených změn a jejich popis:

- Úprava šablony pro HTML export
- Prioritní snapping
- Kopírování prvků Ctrl+tažením

Tyto změny byly navrženy vedoucím práce po zkušenostech s programem Pencil i jinými nástroji pro návrh uživatelských rozhraní a nástroji na tvorbu grafiky obecně. Jelikož se jedná o vylepšení stávajících funkcí programu, je možné jejich budoucí začlenění do standardní distribuce Pencilu.

5.1.1 Úprava šablony pro HTML export

Standardní verze Pencilu obsahuje šablonu pro export do HTML nazvanou GUI Prototyping, jejíž HTML kód však obsahoval chyby vedoucí k nesprávnému zobrazení v internetových prohlížečích. HTML kód bylo nutné z větší části přepsat a provést drobné změny designu, aby bylo zajištěno korektní zobrazení v různých prohlížečích i velikostech okna.

Použití knihovny je následovné:

1. Zvolte *Tools* - *Manage Export Template* - *Install New Template*
2. Vyberte soubor .zip se šablonou
3. Otevřete projekt k exportu
4. Zvolte *File* - *Export Document* - *Single web page*
5. Zvolte rozsah exportu

6. Vyberte šablonu `GUI Prototyping mod`, cestu k exportovanému projektu a stiskněte *Finish*

7. Otevřete soubor `index.html` ve zvoleném adresáři

5.1.2 Prioritní snapping

Pencil v základu obsahuje tzv. *Snapping*. Snapping funguje na principu přichytávání objektů k x, resp. y souřadnicím dalších objektů během tažení myší. Toto chování usnadňuje zarovnávání objektů k sobě. U každého objektu jsou pro snapping v jedné ose k dispozici různé souřadnice. Nejčastěji to jsou horní a spodní hrana objektu v horizontální ose, resp. jeho levý a pravý okraj ve vertikální ose. Souřadnice středu objektu jsou zahrnuty v obou osách. Tento mechanismus je běžně používaný v nástrojích pro práci s grafikou, ve standardní verzi Pencilu jsou však některé nedostatky a byly proto navrženy změny k jejich odstranění.

První vylepšení snappingu spočívalo v zahrnutí souřadnic okna editoru při výpočtu snappingu. Pencil z nějakého důvodu tato data nezahrnoval, tato drobná modifikace tak umožnila přichycení objektu ke krajům okna. To je výhodné např. při roztahování objektu na celou plochu okna nebo při centrování prvků na střed okna.

Dalším vylepšením bylo vyladění rozhodovací úrovně, kdy se má ještě tažený objekt přichytit. Rozhodovací úroveň je v Pencilu standardně nastavena příliš nízko, pro přichycení objektu tak bylo dříve nutné jej dotáhnout až téměř přesně na požadované místo. Vylepšení v podobě zvýšení rozhodovací úrovně po pečlivém otestování vedlo k plynulejší práci s programem během návrhu.

Nejvýznamnější změnou funkcionality snappingu však byla implementace prioritního snappingu, tedy principu, kdy se objekt spíše přichytí k objektu stejného typu. Tato funkce velice usnadňuje práci, pokud se v okně nachází velké množství prvků na různých souřadnicích. Standardní chování při snappingu je pak trhané, jelikož program neví, ke kterému prvku chce uživatel snapping provést. Má implementace prioritního snappingu vychází z předpokladu, že prvky stejného typu mají být pravděpodobně zarovnané k sobě spíše, než objekty různé, např. tlačítko bude většinou zarovnáno k jinému tlačítku spíše, než k textovému prvku nebo čtverci. Algoritmus pro snapping primárně hledá přichycení k objektům stejného typu, s nepartně vyšší rozhodovací úrovní než při běžném postupu. Teprve když není vyhodnocena žádná shoda, hledá se snapping standardním způsobem. Tím je docíleno toho, že pro objekty stejného typu je snapping nalezen přednostně.

5.1.3 Kopírování prvků Ctrl + tažením

Urychlené kopírovaní objektu pomocí tažení se stisknutou klávesou Ctrl je standardní součástí většiny nástrojů pro práci s grafikou i textových editorů a je s podivem, že Pencil zatím tuto funkci neobsahoval. Objekt bylo nutné nejprve zkopírovat do schránky, poté jej ze schránky vložit do okna editoru, a teprve poté přesouvat. Doprogramování této funkce do programu tak výrazným způsobem urychlilo vytváření opakujících se prvků na stránce, jako jsou např. řádky formuláře, tlačítka apod. Tato nová funkcionalita se nijak neliší od chování jiných editorů, je tedy možné tímto způsobem kopírovat i více vybraných objektů naráz.

Pro aktivaci této funkcionality byla zvolena klávesa Ctrl právě proto, že je k tomuto účelu používaná prakticky ve všech nástrojích, které toto chování umožňují. Tato forma konzistence

zajišťuje, že nový uživatel programu funcionalitu snadno objeví a intuitivně začne používat, pokud už s ní má předchozí zkušenosti z jiných aplikací.

5.2 Napojení na libimap

Jakmile byla vytvořena a otestována knihovna `libimap`, a vytvořen kontroler s potřebnými funkcemi, mohl být napojen na funkcionalitu Pencilu. Kód se pro přehlednost nachází ve struktuře zdrojových souborů v adresáři *ExtBrain*. Ten má následující hierarchii:

```
ExtBrain/
   Connectors/
       Css/
   libimap/
       Controller/
       Core/
       Widget/
```

K propojení musely být vytvořeny nebo modifikovány metody některých objektů Pencilu. Tyto metody se nacházejí v adresáři *Connectors*. Princip tohoto obohacení výchozích objektů Pencilu spočívá v rozšíření prototypu existující metody o metodu s jiným názvem. Metody tak nemusí být ve stejném souboru, stačí jen z místa jejich použití importovat nejprve originální objekt a poté jeho rozšíření z adresáře *Connectors*.

5.2.1 Ukládání nastavení

Z důvodů uživatelské přívětivosti jsou do globálních nastavení aplikace ukládána tato nastavení, a to hned po jejich prvním vyplnění:

- Adresa serveru
- Použitý port
- Přepínač použití SSL módu
- Uživatelské jméno účtu na serveru
- Heslo k účtu na serveru
- Email uživatele (pro jeho identifikaci na serveru)

Ve výsledku je tedy uživatelský vstup požadován pouze během prvního připojení k serveru, poté již probíhá plně automaticky. Všechny tyto volby je však možné změnit v nastavení aplikace volbou *Settings* - *Options* pod záložkou *IMAP*.

5.3 Scénáře

Použití knihovny `libimap` v programu Pencil bylo :

- Uložení souboru na server
- Stažení souboru ze serveru

Pro práci klienta byly tedy vytvořeny komponenty pro provedení těchto operací *ImapLoadFile*, resp. *ImapSaveFile*. Vstupním bodem z metod objektů Pencilu do komponent knihovny `libimap` jsou takto pojmenované soubory v jazyce XUL, které definují frontu úkolů (viz 4.2.4.2) potřebných k provedení patřičné operace. Operace je spuštěna otevřením jednoho z těchto souborů metodou `window.openDialog()` a předání konfiguračního objektu jako parametru, viz následující příklad uložení souboru z metody kontroleru:

```
var confObj = {
    fileType : IMAPConfig.tag.project,
    name : this.doc.properties.name,
    filepath : this.filePath,
    msgId : msgId,
    oldMsgId : oldMessageId,
    text: IMAPConfig.tag.project + ": " + this.doc.properties.name,
    out : null
};
window.openDialog("../ExtBrain/libimap/Controller/ImapSaveFile.xul",
        "Save IMAP Document", "chrome,dialog, modal", confObj);
```

Obě tyto operace rozepsány do tzv. scénářů jsou uvedeny níže. Tyto scénáře a popisujících sekvenci provedených akcí:

5.3.1 Uložení souboru na server

1. Připojení na server
2. Autentizace uživatele
3. Získání seznamu složek
4. Vybrání příslušné složky
5. Získání seznamu zpráv
6. Výběr zvolené zprávy
7. Stažení vybrané zprávy
8. Uložení přílohy zprávy na disk
9. Odpojení se od serveru

5.3.2 Stažení souboru ze serveru

1. Připojení na server
2. Autentizace uživatele
3. Získání seznamu složek
4. Vybrání příslušné složky
5. Načtení souboru z disku
6. Formování zprávy
7. Uložení zprávy na server
8. Odpojení se od serveru

5.4 Synchronizace zdrojových souborů

5.4.1 Princip a použití

Stažení a otevření nového souboru ze serveru probíhá volbou *File* - *Open from IMAP*. Klient se poté připojí k serveru a nabídne uživateli seznam projektů k načtení. Po zvolení jednoho z nich je tento stažen, otevřen a připraven k práci. Se souborem se dá manipulovat stejně jako s obyčejnými zdrojovými soubory Pencilu, je možné si ho tedy např. uložit lokálně pro pozdější práci.

Uložení otvřeného projektu z IMAPu, stejně jako první upload nového projektu, se provádí volbou *File* - *Save to IMAP*. Program pak automaticky nahraje projekt na server. Pokud byl projekt otevřen z IMAPu, je během uploadu archivována jeho stará verze.

Při prvním uploadu projektu na server je nutné zjistit, jaký název se má pro uložený projekt použít. V případě nového projektu, který ještě nebyl uložen je uživatel dotázán na název projektu, jinak je tento název odvozen ze jména souboru.

5.4.2 Implementace

Funkcionalita byla napojena do hlavního kontroleru Pencilu. Stejná komponenta zprostředkovává i ukládání a načítání zdrojových souborů lokálně na disk, umístění bodu napojení je tedy správné. Byly implementovány metody `saveIMAPDocument` a `loadIMAPDocument`, které ve spojení s patřičnými scénáři, viz 5.3, ovládají kontroler a IMAP klienta.

5.5 Import a export privátních kolekcí

5.5.1 Princip a použití

Použití funkcionality je konzistentní s přístupem Pencilu k privátním kolekcím obecně, tedy ze záložky *My Stuffs* v paletě nástrojů v levé části okna. Kliknutí pravého tlačítka v

tomto okně vyvolá kontextové menu s volbami pro správu privátních kolekcí v programu Pencil.

Volba *Import new private collection from IMAP* se připojí k serveru a nabídne uživateli seznam dostupných privátních kolekcí. Pokud si uživatel nějakou zvolí, je stažena a nainstalována stejným způsobem jako lokální.

Druhá volba související s IMAPem je *Export this collection to IMAP*. Tato volba lze použít ve stejném případě jako běžný export, tedy pouze pokud je kontextové menu vyvoláno nad existující kolekcí. Po stisknutí je exportovaná kolekce uploadována na server.

5.5.2 Implementace

Tato funcionalita byla napojena do manažeru privátních kolekcí Pencilu, která obaluje všechny metody pro práci s privátními kolekcemi. Pro ovládání této funkcionality vznikly dvě metody, a sice `importNewIMAPCollection` a `exportIMAPCollection`. V rámci druhé zmíněné však byla provedena ještě drobná modifikace funkce Pencilu, a sice refaktorizace části původní metody `exportCollection` do metody `_exportColectionToFile`. Jedná se o společnou část, kterou nyní mohou využívat obě exportní metody a nedochází tak ke zbytečné duplikaci kódu.

Kapitola 6

Testování

Aplikace byla během vývoje průběžně testována, a to jak na funkcionalitu, tak i použitelnost uživatelského rozhraní.

6.1 Pencil

Vzhledem k tomu, že Pencil jako aplikace byl již jeho vývojáři vyladěn, tato část testování byla věnována uživatelskému rozhraní propojení programu s klientem protokolu IMAP. Práce s tímto klientem by však měla být co nejvíce automatická a uživatele by neměla rušit od práce. Z tohoto důvodu jsou veškerá uživatelská nastavení knihovny po prvním zadání uložena do globálních preferencí programu a práce pak probíhá bez nutnosti uživatelského vstupu.

Přesto se během testování s uživateli objevily některé nedostatky. První verzi byla vytknuta přílišná prototypovost a byly podniknuty kroky k jejímu vylepšení. Byly např. odstraněny prvky typu `prompt`, přijímající textový vstup od uživatele, a jako jejich náhrada byly implementovány vlastní kontrolní prvky. Druhým příkladem je, že uživatel byl původně dotazován na specifikaci serveru a přihlašovací údaje ve dvou krocích. Tyto dva kroky byly kvůli lepší přehlednosti a zrychlení celého procesu sloučeny do jednoho.

Další změna se týká začištění chybových hlášek programu. Ty jsou v některých případech bohužel nutné, podařilo se ale zjednodušit např. autentikaci uživatele v případě, že zadá špatné přihlašovací údaje. Pokud server po prvním špatném zadání odmítne uživatele znovu autentizovat, dojde automaticky k odpojení a znovupřipojení bez nutnosti uživatelské akce.

6.2 libimap

Testování knihovny `libimap` muselo být o mnoho rozsáhlejší než testování několika případů uživatelského vstupu v Pencilu.

Asi nejvýznamnější nahlášeným problémem bylo (naštěstí hned v začátku vývoje) mrznoucí uživatelské rozhraní během čekání na odpověď. To bylo způsobeno synchronní implementací komunikace se serverem, program neodpovídal po celou dobu co klient čekal na zprávu od serveru. Kvůli této chybě byla komunikace přepsána asynchronně, což se ukázalo jako správné rozhodnutí.

Dalším nedostatkem byla absence timeoutu při pokusu o přípojení se k serveru. I tato funkce však již byla do programu doplněna.

6.2.1 Serverové implementace IMAPu

Rozsáhlé testování muselo také proběhnout ohledně použitelnosti knihovny na běžně používaných serverech podporujících IMAP. Testovány byly serverové implementace emailových komponent následujících serverů:

- gmail.com
- feld.cvut.cz
- aol.com
- centrum.cz
- e-fractal.cz
- yahoo.com

Tato fáze testování velmi pomohla vylepšit implementaci klienta, protože jednotlivé systémy se od sebe liší. To je dáno faktem, že specifikace RFC3501[1] není příliš striktní a nechává volbu některých chování na rozhodnutí vývojářů. To samozřejmě přidává práci vývojářům klientských aplikací, jelikož musí své programy přizpůsobit všem z nich.

Příkladem takto rozdílného chování je např. podpora ukládání vlastních flagů (viz 3.3.4) u zpráv. První dva jmenované (*gmail.com* a *feld.cvut.cz*) plně podporují přidávání těchto flagů ke zprávám při jejich nahrávání na server, tak jak to vyplývá z RFC3501. Zároveň je samozřejmě možné s nimi manipulovat pomocí příkazu `STORE`.

Firemní server *e-fractal.cz* s implementací IMAPu `Kerio Connect`[4] a free mail služba na *yahoo.com* vlastní flagy nepodporují a jsou nejstriktnější ze všech jmenovaných, protože pokud jim v rámci příkazu `APPEND` přijde neznámý flag, celý příkaz selže a celá zpráva přenesená na server zanikne. Server se však alespoň chová konzistentně i při použití příkazu `STORE`, který při použití vlastního flagu selže také.

Naproti tomu server *centrum.cz* sice tyto tagy také nepodporuje, ale jeho přístup mi osobně přijde lepší. Pokud serveru přijde požadavek na vlastní flag, je tento flag ignorován, zbytek příkazu ale proběhne dobře. To platí jak pro příkaz `APPEND`, tak i `STORE`. Nevýhodou samozřejmě je, že klient s takovýmto postupem musí počítat a nečekat přítomnost flagů u zpráv.

Server *aol.com* je specifický ještě více, protože flagy sice ignoruje a výsledným chováním odpovídá předchozím dvěma, ale navíc, při pokusu o dodatečné vytvoření vlastního flagu voláním příkazu `STORE` odpoví, jako by vytvoření tohoto flagu dopadlo úspěšně. Navíc, přestože tento flag nelze nijak používat, objeví se mezi ostatními dostupnými flagy, což ještě více poukazuje na nesmyslnost návrhu této serverové implementace.

Program musel být modifikován, aby zahrnoval mechanismus pro všechny tyto případy a umožnil alespoň omezené použití i nepodporovaných serverů. Nyní podle podpory uživatelských flagů na serveru zvolí pro připojení jeden ze dvou módů. První z nich funguje tak jak

bylo zamýšleno a popsáno, zatímco druhý vynechává vytváření uživatelských flagů při ukládání projektů na server. Ze stejného důvodu není možné filtrovat obsah složky právě podle typu, proto výběr souboru pro otevření projektu i import privátní kolekce zobrazí v druhém módu oba typy. Uživatel je na provoz ve druhém (nepodporovaném) módu upozorněn při prvním připojení.

6.3 Testovací účty

Pro vyzkoušení této práce následuje přehled serverů, na kterých bylo prováděno testování programu a přihlašovací údaje k nim. Z bezpečnostních důvodů samozřejmě není možné poskytnout uživatelská jména a hesla k osobním účtům, kde to však provozovatel umožnil, tam byly vytvořeny účty speciálně pro testování.

Adresa serveru	Port	Typ	Uživatelské jméno	Heslo	Implementace
imap.gmail.com	993	ssl	test.imap.cz	testimap	Gimap
imap.feld.cvut.cz	993	ssl	...	...	Cyrus IMAP v2.4.12
imap.aol.com	993	ssl	test.imap.cz	account	
imap.centrum.cz	993	ssl	test.imap.cz	testimap	balIMap
mail.e-fractal.cz	993	ssl	...	...	Kerio Connect
imap.mail.yahoo.com	993	ssl	test.imapcz	testimap	

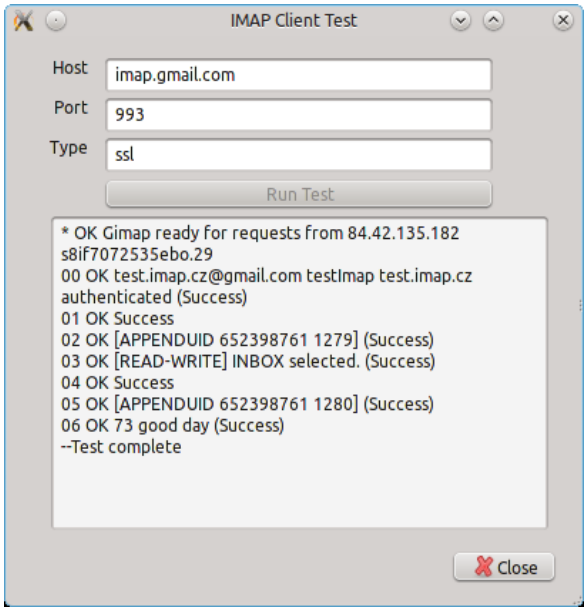
Tabulka 6.1: Přehled testovaných serverů

6.4 Testovací aplikace

Souběžně s vývojem knihovny `libimap` byla vyvíjena i aplikace s testovacím scénářem. Jejím úkolem je vyzkoušet mechanismy a příkazy knihovny na testovacích datech. Po každé modifikaci bylo tímto testem ověřeno zachování funkčnosti aplikace jako celku. V praxi je tato praktika nazývána *Test Driven Development*[7]

Jelikož vývoj knihovny byl problematický a zdrojový kód se v průběhu implementace několikrát změnil v podstatě od základů, tento soubor testů se ukázal jako neocenitelný.

Na obrázku 6.1 je screenshot okna aplikace testovacího scénáře knihovny `libimap`.



Obrázek 6.1: Snímek okna testovací aplikace

Kapitola 7

Další vývoj

Jelikož tato bakalářská práce je přímo ze zadání určena k dalšímu vývoji, je celá tato kapitola věnována návrhu možných budoucích funkcí především dvou hlavních komponent - programu Pencil a JavaScriptové knihovny libimap. Tyto návrhy vznikly během analýzy této práce, z důvodu jejich velkého rozsahu a náročnosti však nemohou být její součástí.

7.1 Pencil

7.1.1 Přímý export UI do HTML

Uživatelské rozhraní vytvořené v Pencilu i ostatních současných nástrojích podobného typu lze exportovat pouze jako obrázek a může tedy sloužit pouze jako prototyp a předloha, který je nutné teprve posléze přepsat do jazyka cílové platformy (např. `HTML`, `XUL`, `Java Swing` nebo grafické toolkity dalších jazyků). Jako funkcionalita by bylo velice výhodné umět exportovat popis `UI` přímo do jazyka této cílové platformy. Jako nejužitečnější z vyjmenovaných možností se nabízí `HTML`, kvůli pravděpodobně nejširší možnosti uplatnění.

7.1.2 Vylepšení vkládání elementů z privátních kolekcí

Při vkládání elementu z privátní kolekce dochází k naklonování a vložení celého tohoto elementu. Lepší variantou by bylo vkládat pouze odkazy na prvek z kolekce. Zřejmou výhodou tohoto přístupu je úspora místa při ukládání souboru, což hraje roli zejména u rozsáhlých projektů.

Druhá výhoda spočívá ve zpětné propagaci změněného elementu do celého dokumentu. V současnosti v Pencilu neexistuje možnost prvek kolekce editovat, pravděpodobně právě z tohoto důvodu. Pokud tedy návrhář potřebuje změnit prvek kolekce, musí ho nejprve vložit do dokumentu, provést změny, smazat z kolekce starou verzi a uložit nový element pod stejným jménem, jako měl ten původní. Poté je potřeba najít všechny výskyty původního prvku a nahradit je novým.

Pokud by však bylo možné prvek změnit a tyto změny by se okamžitě propagovaly do všech výskytů tohoto prvku v dokumentu, došlo by k dalšímu zrychlení procesu návrhu. Celé funkcionalita privátních kolekcí by tímto získala na použitelnosti, zejména pak u velkých dokumentů nebo prvků opakujících se ve více projektech.

7.1.3 Layoutovací principy

Důležitou součástí popisu uživatelského rozhraní je popis rozmístění prvků na stránce, tzv. layout. Pencil v současnosti umožňuje pouze práci s absolutním pozicováním. Během vytváření je sice k dispozici alespoň snapping, ten ale stejně slouží pouze ke zjištění absolutní pozice pro element. Existují čtyři základní layoutovací principy. Kterýkoliv z nich by byl pro návrh UI obrovským přínosem, ale stejně tak vyžaduje implementace kteréhokoliv z nich kompletní změnu práce s daty. Následuje přehled těchto layoutovacích principů, k prvním třem z nich je uveden příklad v jazyce `HTML` s použitím kaskádových stylů `CSS`. Zdůvodnění volby `HTML` viz. 7.1.1

- stacking
- wrapping
- anchoring
- docking

7.1.3.1 Stacking

Pojmem stacking se rozumí vkládání prvků pod sebe do kontejneru.

Stacking je jedním ze základních principů umisťování elementů do `HTML` stránky. Všechny elementy ve stejném kontejneru jsou zobrazeny pod sebou v pořadí, v jakém jsou definovány ve zdrojovém kódu. Stacking je aktivován nastavením stylu elementu `display: block`. Některé elementy mají tuto vlastnost nastavenu jako výchozí, např. základní stavební jednotka webových stránek - element `div`. Pokud součet výšek všech elementů překročí výšku kontejneru, ten se může buď začít roztahovat nebo nechat elementy podtéct a skrýt je. Tohoto je dosaženo nastavením css atributu `overflow: hidden`.

```
<div></div>
<div></div>
```

7.1.3.2 Wrapping

Anglický pojem wrapping označuje princip, kdy jsou prvky vkládány vedle sebe, dokud součet jejich šířek nepřekračuje vnitřní šířku kontejneru, ve kterém se nachází. Poté dochází k zalomení řádku a další prvky jsou dále vkládány za sebe tam.

Vedle stackingu je toto druhým základním principem vkládání obsahu do stránky. Nastavení wrappinu objektu probíhá nastavením `display:inline`, resp. `display:inline-block`. Mezi elementy s touto vlastností nastavenou jako výchozí patří `span`, `a`, `img`.

Rozšíření možností wrappingu v `HTML` je navíc možné použitím `CSS` stylu `float`. Jeho podpora se však v různých zobrazovacích enginech liší.

```
.floatingElement {
    width:50px;
    float:left;
}
```

7.1.3.3 Anchoring

Ukotvení nebo pozicování elementu, tzv. anchoring je pravděpodobně nejpoužívanější možnosti popisu zobrazení elementu v grafickém rozhraní. Element má přesně zadané souřadnice na kterých se má zobrazit. Tyto souřadnice jsou zadány v libovolných jednotkách, které renderovací engine podporuje. Ukotvení je používané ve většině návrhářů `GUI`, ať už se jedná o součásti vývojových nástrojů pro programovací jazyky nebo prototypovací nástroje. Princip ukotvení používají i programy pro práci s grafikou, kde jsou elementy umisťovány na kreslící plochu uživatelem právě pomocí souřadnic.

Anchoring je v platformě `HTML` prováděn definováním absolutní pozice elementu uvnitř kontejneru. Šířka elementu pak může být předepsána buď relativně ke kontejneru pozicí levého a pravého okraje prvku, nebo přímo. Obdobně pak lze předepsat výšku elementu buď přímo, nebo pozicí horního a spodního okraje, viz příklad.

```
.anchoredElement {
    position:absolute;
    top:10px;
    bottom:50px;
    left:15px;
    right:15px;
}
```

Jak už bylo zmíněno, Pencil v současnosti pro umístění elementů pozicování používá. Jedná se ale pouze o absolutní pozicování směrem k levému hornímu okraji dokumentu, chybí ostatní možnosti pozicování jako v `HTML`.

7.1.3.4 Docking

Zarovnávání panelů zvané docking (česky dokování), je vlastně spojením všech tří předchozích principů. Dokované elementy jsou zarovnávány k okrajům kontejneru.

Platforma `HTML` nepodporuje žádnou nativní formu dokování, lze však docílit stejného zobrazení prvků použitím ostatních principů. Není však možné provádět docking automaticky a je tedy potřeba předchozí znalost toho, jak má výsledek vypadat.

7.2 libimap

7.2.1 Kompletní implementace standardu

Implementovaná knihovna `IMAP` klienta `libimap` zatím neobsahuje všechny příkazy vyplývající ze specifikace RFC3501[1]. Implementovány byly pouze příkazy potřebné v této práci, jedná se však o všechny příkazy potřebné pro základní komunikaci s `IMAP` servery.

S implementací zbývajících příkazů pak souvisí i dokončení zbytku knihovny tak, aby přesně odpovídala specifikaci RFC3501[1].

7.2.2 Příkaz IDLE

`IDLE` je nadstandartní příkaz protokolu `IMAP` a je definován v RFC2177[5]. Při standartním přístupu k serveru je potřeba, aby se klient na stav složky vždy dotázal. Naproti tomu použití příkazu `IDLE` umožňuje přijímat od serveru notifikace v reálném čase. V praxi příkaz funguje tak, že klient po volání příkazu „naslouchá“ a server posílá upozornění na změny ve vybrané složce, jakmile dojde ve složce k nějaké změně, tedy např. když přijde nová zpráva nebo se složkou manipuluje jiný klient.

Využití tohoto příkazu přináší klientovi neustálou kontrolu nad obsahem složky bez nutnosti se neustále dotazovat. To ušetří množství přenesených dat, zbytečných requestů na server i času, jelikož klient je na všechny změny na serveru upozorněn téměř v reálném čase.

7.2.3 Platformní nezávislost

Pro potřeby této práce je knihovna implementována v Mozilla Application Frameworku a využívá tedy komponent této platformy. V důsledku lze tedy použít pouze v programech postavených taktéž na MAF, např. tedy i ve webovém prohlížeči Firefox. Pokud by však mohla být knihovna platformně nezávislá, bylo by možné s ní pracovat ve všech prohlížečích a značně by se rozšířily možnosti jejího využití. bylo by tak možné vytvořit s její pomocí např. univerzálního webového klienta elektronické pošty. Zároveň by bylo možné udělat z primitivního nástroje `libimap-console`, viz 4.6, plnohodnotnou aplikaci nezávislou na prohlížeci Mozilla Firefox.

Při vývoji byl na tuto možnost rozšíření brán zřetel a platformně závislé funkce a volání byly v kódu aplikace sloučeny pod `MozExtbrainToolkit`. Pokud bude možné vytvořit platformně nezávislý `XPExtbrainToolkit` implementující stejné chování, bude možné jej bez problémů napojit na knihovnu `libimap`.

7.3 MozTextSocketConnection

Při návrhu a implementaci bylo dbáno na správné rozdělení kódu do tříd tak, aby byla zajištěna co největší modularita. V předchozí sekci již bylo řečeno, že knihovna `libimap` může fungovat s každou patřičnou třídou pro připojení k serveru.

Stejným způsobem je možné využít současnou třídu pro zprostředkování komunikace, třídu `MozTextSocketConnection` pro jakoukoliv jinou komponentu, která potřebuje komunikovat po síti a je pro ni dostupný Mozilla XPCOM. `MozTextSocketConnection` byla totiž navržena tak, aby zprostředkovávala pouze textovou komunikaci se serverem a neobsahovala žádnou funkcionalitu specifickou pro protokol `IMAP`. Právě proto může být použita i platformně nezávislá třída pro připojení z předchozí sekce, jakmile bude implementována.

Kapitola 8

Závěr

V této bakalářské práci se podařilo využít existující návrhář uživatelských rozhraní Pencil a rozšířit ho o možnost synchronizovat vytvořené projekty na server podporující poštovní protokol IMAP. Na vzdálené úložiště lze také ukládat i privátní kolekce uživatelských objektů a načítat je odtud. Nová funkcionalita přináší řadu výhod, např. zálohování práce mimo počítač nebo významné urychlení spolupráce na komplexnějších projektech. Výstupem této práce je tedy kompletní program pro návrh UI, použitelný zároveň ke kolaboraci na rozsáhlých uživatelských rozhraních.

Zároveň byla v rámci této bakalářské práce vytvořena knihovna pro připojení k serverům IMAP. Jedná se o funkční implementaci klientské části protokolu IMAP a její součástí je i vzorová mini-aplikace, určená jako tutorial pro vývojáře, kteří budou knihovnu chtít používat ve vlastních aplikacích. V době odevzdání této práce dokonce knihovnu již používá Bc. David Kalivoda v rámci své diplomové práce na téma „Synchronizace kontaktů pro ExtBrain Communicator“, taktéž pod vedením Ing. Novotného.

Knihovna poslouží vývojářům nejen v rámci projektu ExtBrain Communicator, ale může být využita prakticky libovolnou aplikací v jazyce JavaScript. Důkazem tohoto tvrzení je i vytvoření mini-aplikace emulující konzolové připojení k IMAP serverům, která sice nesouvisí se zadáním práce, ale významně ulehčila její ladění. Stejným způsobem může být užitečná i ostatním vývojářům, kteří si potřebují ověřit chování serverových implementací IMAPu.

Síťová část programu Pencil i zdrojový kód knihovny byly zdokumentovány, z důvodu co největšího usnadnění práce budoucích vývojářů.

Možností budoucího rozvoje této práce byly již vyjmenovány v kapitole 7. Jeden z těchto návrhů je dokonce již těsně před dokončením, a sice odstranění platformní závislosti knihovny `libimap` na platformě Mozilla. To je pro budoucí použití knihovny zásadní krok, který významně rozšiřuje možnosti jejího použití.

Nezanedbatelným výsledkem práce je i osobní přínos pro mě. Samostatný projekt takovéhoto rozsahu i trvání pro mě byl absolutní novinkou, a v průběhu práce na něm jsem se mnohému naučil. Objevil jsem obrovské možnosti programování v JavaScriptu a naučil se nové techniky velmi cenné v praxi. Z analytického i programátorského hlediska pro mě byla práce velice užitečná.

Literatura

[1] CRISPIN, M. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1* [online]. 2003. [cit. 2. 1. 2012]. Dostupné z: <http://tools.ietf.org/html/rfc3501>.

[2] EVOLUS. *Pencil* [online]. 2010. [cit. 2. 1. 2012]. Dostupné z: <http://pencil.evolus.vn/en-US/Home.aspx>.

[3] GALITZ, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques.* : John Wiley & Sons, 3rd edition, 2007. In English.

[4] KERIO. *Kerio Connect* [online]. 2011. [cit. 3. 1. 2012]. Dostupné z: <http://www.kerio.com/connect>.

[5] LEIBA, B. *IMAP4 IDLE command* [online]. 1997. [cit. 2. 1. 2012]. Dostupné z: <http://tools.ietf.org/html/rfc2177>.

[6] LEIBA, B. *IMAP4 Implementation Recommendations* [online]. 1999. [cit. 4. 1. 2012]. Dostupné z: <http://tools.ietf.org/html/rfc2683>.

[7] NETWORK, M. D. *Introduction to Test Driven Development (TDD)* [online]. 2003. [cit. 3. 1. 2012]. Dostupné z: <http://www.agiledata.org/essays/tdd.html>.

[8] NETWORK, M. D. *Mozilla Thunderbird* [online]. 2011. [cit. 3. 1. 2012]. Dostupné z: <http://www.mozilla.org/projects/thunderbird/>.

[9] NETWORK, M. D. *XPCOM* [online]. 2011. [cit. 3. 1. 2012]. Dostupné z: <https://developer.mozilla.org/en/XPCOM>.

[10] NOVOTNÝ, T. *ExtBrain* [online]. 2011. [cit. 2. 1. 2012]. Dostupné z: <http://extbrain.felk.cvut.cz/>.

[11] PROJECT, T. O. *OpenSSL* [online]. 2011. [cit. 3. 1. 2012]. Dostupné z: <http://www.openssl.org>.

[12] RESNICK, P. *Internet Message Format* [online]. 2001. [cit. 2. 1. 2012]. Dostupné z: <http://tools.ietf.org/html/rfc2822>.

[13] SZEKELY, P. *User Interface Prototyping: Tools and Techniques* [online]. 1994. [cit. 3. 1. 2012]. Dostupné z: <http://www.isi.edu/~szekely/contents/papers/1994/ui-prototyping-ICSE-1994.pdf>.

[14] TATHAM, S. *PuTTy* [online]. 2011. [cit. 3. 1. 2012]. Dostupné z: <http://www.putty.org>.

Příloha A

Seznam použitých zkratek

Následuje seznam zkratek použitých v této práci. Seznam je v abecedním pořadí

AJAX Asynchronous JavaScript and XML

CRLF Carriage Return and Line Feed

CSS Cascade Style Sheet

GUI Graphical User Interface, uživatelské rozhraní

GNU GPL GNU General Public License

HTML HyperText Markup Language

IMAP Internet Message Access Protocol

MAF Mozilla Application Framework

POP Post Office Protocol

UI User Interface

SSL Secure Socket Layer

XHTML eXtensible HyperText Markup Language

XML eXtensible Markup Language

Příloha B

Příklady komunikace s protokolem IMAP

B.1 Příkaz APPEND

Následuje příklad komunikace mezi klientem a serverem během použití příkazu `APPEND`. Jedná se o zachycený záznam komunikace, testovací data jsou převzata z RFC3501.

```
C: 01 APPEND INBOX (\Seen) {310}
S: + go ahead
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
From: Fred Foobar <foobar@blt.example.COM>
Subject: afternoon meeting
To: test.imap.cz@gmail.com
Message-Id: <B27397-0100000@blt.example.COM>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII

Hello Joe, do you think we can meet at 3:30 tomorrow?


S: 01 OK [APPENDUID 652398761 5] (Success)
```

B.2 Příkaz LIST

V dalším příkladu je zobrazeno použití příkazu `LIST` a jeho parametrů.
Znak hvězdička "*"je tzv. "wildcard"a odpovídá libovolnému počtu libovolných znaků v názvu složky na jeho pozici.

```
C: 03 list "" "*"
S: * LIST (\HasNoChildren) "/" "INBOX"
* LIST (\Noselect \HasChildren) "/" "[Gmail]"
```

```
* LIST (\HasNoChildren) "/" "[Gmail]/All Mail"
* LIST (\HasNoChildren) "/" "[Gmail]/Drafts"
* LIST (\HasNoChildren) "/" "[Gmail]/Important"
* LIST (\HasChildren \HasNoChildren) "/" "[Gmail]/Sent Mail"
* LIST (\HasNoChildren) "/" "[Gmail]/Spam"
* LIST (\HasNoChildren) "/" "[Gmail]/Starred"
* LIST (\HasChildren) "/" "[Gmail]/Trash"
* LIST (\HasNoChildren) "/" "backup"
* LIST (\HasNoChildren) "/" "new"
* LIST (\HasNoChildren) "/" "test"
03 OK Success
```

Znak procento "%"je další "wildcard"podobný znaku hvězdička "*", ale odpodívá všem znakům kromě znaku oddělovače v hierarchii složek (v následujícím příkladu je v serverové implementaci tímto oddělovačem znak lomítko "/").

```
C: 04 list "" "%"
S: * LIST (\HasNoChildren) "/" "INBOX"
* LIST (\Noselect \HasChildren) "/" "[Gmail]"
* LIST (\HasNoChildren) "/" "backup"
* LIST (\HasNoChildren) "/" "new"
* LIST (\HasNoChildren) "/" "test"
04 OK Success
```

B.3 Kompletní komunikace

Následující výpis zobrazuje jednoduchý příklad komunikace mezi klientem a serverem. Ta byla pro ilustrační účely mírně modifikována a rozdělena na samostatně vysvětlené bloky.

```
* OK Gimap ready for requests from 82.100.0.156 3if4188869fak.68
```

Úvodní zpráva přichází od serveru po navázání spojení. Hvězdička na začátku označuje netagovanou zprávu, viz. 3.3.2

```
00 login test.imap.cz testimap

* CAPABILITY IMAP4rev1 UNSELECT IDLE NAMESPACE QUOTA ID XLIST
CHILDREN X-GM-EXT-1 UIDPLUS STARTTLS

00 OK test.imap.cz@gmail.com testImap test.imap.cz authenticated (Success)
```

Klient se poté přihlásí příkazem login se svým uživatelským jménem a heslem, od serveru obdrží výpis schopností serveru a s ním i kladnou odpověď na požadavek.

```
01 list "" %
* LIST (\HasNoChildren) "/" "INBOX"
```

```
* LIST (\HasNoChildren) "/" "Personal"
* LIST (\HasNoChildren) "/" "Work"
* LIST (\Noselect \HasChildren) "/" "[Gmail]"
01 OK Success
```

Klient zašle serveru požadavek na vylistování všech složek v kořenovém adresáři. S odpovědí přijde i seznam složek. Důležitý je v tuto chvíli třetí parametr odpovědi, a sice jména složek (`INBOX`, `Personal`, `Work`).

```
02 select INBOX
* FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
* OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen \*)]
* OK [UIDVALIDITY 652398761]
* 4 EXISTS
* 0 RECENT
* OK [UIDNEXT 5]
02 OK [READ-WRITE] INBOX selected. (Success)
```

Následujícím příkazem klient odešle požadavek ke zvolení jedné z nich, složky `INBOX`. Kromě kladné odpovědi serveru je mu vrácen ještě přehled o obsahu složky. Mezi důležité údaje zde patří celkový počet zpráv ve složce (`EXISTS`). Od této chvíle je složka `INBOX` označena jako zvolená pro další práci se zprávami.

```
03 FETCH 4 BODY.PEEK[HEADER.FIELDS (DATE SUBJECT FROM)]
* 4 FETCH (BODY[HEADER.FIELDS (DATE SUBJECT FROM)] {101}
From: Petr Mach <pe.mach@gmail.com>
Date: Sat, 14 May 2011 13:52:08 +0200
Subject: Test message

)

03 OK Success
```

Příkazem `FETCH` klient stahuje zprávu ze serveru. Tvar příkazu je podrobně popsán výše. V tomto případě klient žádá o náhled na data zprávy číslo 4, konkrétně výběr dat z její hlavičky - odesílatele, datum a předmět zprávy.

```
04 LOGOUT

* BYE LOGOUT Requested

04 OK 73 good day (Success)
```

Když už klient dále připojení k serveru nepotřebuje, musí se před uzavíráním komunikačního kanálu ze serveru odhlásit příkazem `LOGOUT`. Teprve po obdržení odpovědi se zprávou `BYE` klient i server ukončí spojení.

Příloha C

Instalační a uživatelská příručka

C.1 Spuštění aplikace

Tato sekce obsahuje informace k postupu spuštění programu.

C.1.1 Windows

C.1.1.1 Pencil

Jelikož je Pencil distribuován jako standalone aplikace, stačí pouze spustit soubor `pencil.exe` z adresáře `exe\pencil`

C.1.1.2 libimap-test

Jelikož testovací mini-aplikace knihovny libimap je určena pouze vývojářům, není pro ní zkompilovaný binarní spustitelný soubor. Spouštění probíhá následovně:

1. Otevřete příkazovou řádku Windows
2. Přesuňte se do adresáře `exe`
3. Spusťte následující příkaz: `xulrunner\xulrunner.exe libimap-test\app\application.ini`

C.1.1.3 libimap-console

Pro spuštění emulátoru konzole otevřete soubor `libimap-console\libimap-console.htm` v prohlížeči Mozilla Firefox. Údaje pro připojení jsou přednastaveny, ale lze použít údaje jakéhokoliv jiného serveru podporujícího protokol IMAP. Stiskněte tlačítko `Connect`, Firefox se nejprve dotáže na povolení k použití `UniversalXPConnect`, poté by server již měl poslat zprávu o otevření spojení.

C.1.2 Linux

Instalace na linuxu je o něco složitější a je potřeba nejprve nainstalovat Xulrunner. Návod lze najít na <https://developer.mozilla.org/en/XULRunner>

C.1.2.1 Pencil

1. Otevřete terminál
2. Přesuňte se do adresáře `exe`
3. Spusťte následující příkaz: `xulrunner pencil/app/application.ini`

C.1.2.2 libimap-test

1. Otevřete terminál
2. Přesuňte se do adresáře `exe`
3. Spusťte následující příkaz: `xulrunner libimap-test/app/application.ini`

C.1.2.3 libimap-console

Otevřete soubor `libimap-console/libimap-console.htm` v prohlížeči Mozilla Firefox. Zbytek instrukcí je totožný s nastavením ve Windows.

C.2 Nastavení serveru IMAP

Během prvního připojení se program dotáže na specifikace serveru a autorizační údaje uživatele. Tyto údaje lze nastavit také ve volbách `Settings - Options - IMAP`. Seznam vytvořených účtů je uveden v části 6.3, pro rychlé nastavení je možné použít následující volby:

- host: imap.gmail.com
- port: 993
- ssl: ano
- username: test.imap.cz
- password: testimap
- user email: Vaše emailová adresa

C.3 Práce s aplikací

Po nastavení IMAPového serveru si volbou *File - Open from IMAP* otevřete projekt ze vzdáleného úložiště. Podmínkou samozřejmě je, že na serveru nějaké projekty existují. Pokud si přejete uložit změněnou verzi zpět, provedete tak volbou *File - Save to IMAP*.

Na stejném pricipu funguje i manipulace s privátními kolekcemi. Ze záložky *My Stuffs* je možné si stáhnout kolekci stisknutím pravého tlačítka myši (v panelu záložky) a volbou *Import new private collection from IMAP*, resp. exportovat kolekci na server volbout *Export this collection to IMAP*.

Příloha D

Obsah přiloženého CD

```
/--- exe                          spustitelné soubory a zdrojové kódy
|    |--- libimap-console
|    |--- libimap-test
|    |--- pencil
|    |--- xulrunner               standalone Xulrunner pro Windows
|    '--- GUI_mod.zip             exportní šablona
|--- src                          pouze zdrojové kódy
|    |--- GUI_mod
|    |--- libimap-console
|    |--- libimap-test
|    |--- pencil
|    '--- thesis                  zdrojové soubory textu práce
|--- text
|    '---  machpet2_2012_bach.pdf text práce ve formátu PDF
|--- install.txt                  postup spouštění software
'--- README.txt                   soubor se strukturou obsahu cd
```