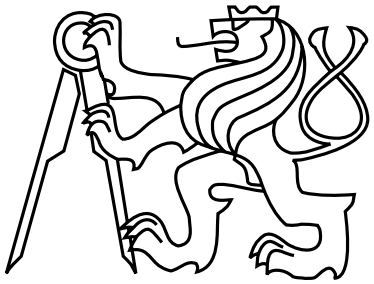
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Integrace Mozilla Thunderbird a XMPP

David Jirovec

Vedoucí práce: Ing. Tomáš Novotný

Studijní program: Elektrotechnika a informatika, strukturovaný, Bakalářský

Obor: Výpočetní technika

8. července 2009

Poděkování

Chtěl bych poděkovat Ing. Tomáši Novotnému za vedení této práce, užitečné rady a nápady.

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 8. 7. 2009 .....

Abstract

The goal of this study is to explore possibilities of Gecko framework, which is used in Mozilla Thunderbird and to implement addon, which allows IM communication over XMPP protocol.

Abstrakt

Tato práce za zabývá průzkumem možností frameworku Gecko, konkrétně aplikací Mozilla Thunderbird a integrací IM komunikace přes protokol XMPP do ní.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Mozilla Thunderbird

Mozilla Thunderbird je jedním z nejznámějších poštovních poštovních klientů současnosti. Stejně jako všechny ostatní Mozilla produkty je vyvíjen jako multiplatformní open source pod trojlicencí GPL/LGPL/MPL a díky použití frameworku Gecko je snadno rozšiřitelný o další funkce. Na oficiálních stránkách je dostupných přes tisíc různých doplňků[1].

1.2 XMPP

XMPP je decentralizovaný otevřený protokol pro IM komunikaci. Zprávy se předávají ve formátu XML. XMPP podporuje kromě komunikace dvou uživatelů i chatovací místnosti, kde může vzájemně komunikovat více uživatelů najednou. Dále stojí za zmínku transporty, které umožňují komunikaci s uživateli z jiných IM protokolů. Nabídka těchto služeb závisí na konkrétním serveru, některé navíc nabízejí služby jako předpověď počasí, televizní program a podobně.

1.3 Proč tito dva?

Díky své otevřenosti a rozšiřitelnosti představuje Thunderbird dobrý základ pro přidání funkce IM klienta a XMPP protokol se zdá jako dobrá volba opět díky otevřenosti a dobré dokumentaci.

Kapitola 2

Popis problému, specifikace cíle

2.1 Cíl projektu

Cílem projektu je vytvořit pro Mozilla Thunderbird doplněk pro IM komunikaci pomocí protokolu XMPP. Uživatelské rozhraní bude inspirováno webovým rozhraním Gmailu, zajímavá by také byla integrace Google Talk pluginu, který by umožil hlasovou komunikaci a video komunikaci.

2.2 Existující řešení

2.2.1 Sameplace

Sameplace[12] je asi nejznámější doplněk přidavající do Thunderbirdu možnost komunikace přes protokol XMPP. Zajímavostí je, že kromě obyčejné textové komunikace přidává další funkce jako hry nebo sdílený kurzor. Okna s kontakty nebo s rozhovory je možné odpojit z hlavního okna Thunderbirdu a používat je zvlášť (samotný Thunderbird ale stále musí běžet). Za zmínku stojí ještě podpora OpenID.

Zároveň se Samplacem se nainstaluje doplněk xmpp4moz, který slouží jako komunikační vrstva pro Sameplace, ale je volně použitelný i pro jiné doplňky.

2.2.2 Instantbird

Instantbird[6] je samostatný IM klient, ale stejně jako Thunderbird je psán v XUL. Zatím je v rané fázi vývoje, oficiálně ještě nejsou k dispozici žádné doplňky, i když podpora pro ně v Instantbirdu je. Klient využívá komunikační knihovnu libpurple, který je základem třeba i známého Pidginu, o kterém je psáno dále. Právě díky libpurple se Instantbird neomezuje pouze na XMPP, ale podporuje i širokou škálu dalších IM protokolů. Díky tomu, že Instantbird je open source pod trojlicencí GPL/LGPL/MPL, by bylo možné využít jeho způsob integrace libpurple do XUL i jinde.

2.2.3 Spicebird

Spicebird[13] je aplikace založená na Thunderbirdu, oproti němu nabízí v základu navíc IM, kalendář a úkolníček. Spicebird nepatří přímo do rodiny Mozilla produktů, je vyvíjen nezávisle Indickou společností Synovel, ale stále je open source pod trojlicencí GPL/LGPL/MPL.

2.2.4 Contacts Sidebar

Contacts Sidebar[3] zobrazuje obsah adresáře v postranním panelu přímo v hlavním okně Thunderbirdu. Tento doplněk nepřináší podporu IM, ale bude zdrojem inspirace pro seznam kontaktů.

2.2.5 Pidgin

Pidgin[11], dříve známý jako Gaim je open source multiplatformní multiprotokolový IM klient pod licencí GPL. Tento program bude opět inspirací, budou z něj využity některé grafické prvky, například ikonky stavů. Díky GPL licenci to není problém. Dále bude používán na testování komunikace.

2.2.6 Gmail

Gmail[5] je zdarma poskytovaná e-mailová služba od společnosti Google. Zajímavostí je, že ve webovém rozhraní je integrovaný IM klient Google Talk s podporou hlasových a video hovorů. Komunikace probíhá protokolem XMPP. E-maily Gmail seskupuje do vláken, takže jejich zpětné procházení připomíná listování v historii IM.

2.3 Použité nástroje

2.3.1 Venkman

Základním vývojovým nástrojem pro tento projekt je Venkman[16], JavaScriptový debugger pro Mozilla aplikace. Je velmi užitečný nejen pro ladění vlastního kódu, ale i pro prozkoumávání kódu samotného Thunderbirdu či jiných doplňků, jelikož práce se stávajícími zdrojovými kódy je pro programátora dosud neznalého implementace Thunderbirdu také skoro nezbytná a velmi poučná.

2.3.2 DOM Inspector

Dalším nezbytným nástrojem je doplněk DOM Inspector[4]. Umožňuje procházení a editaci stromu Document Object Modelu HTML nebo XUL dokumentů. Výběr uzlu je možný buď ve výše zmíněném stromu, nebo přímo kliknutím na prvek dokumentu. Možná zobrazení uzlu jsou DOM Node, Box Model, XBL Bindings, CSS Style Rules, Computed Style a JavaScript Object.

2.3.3 SQLite Manager

Protože vyvíjený doplněk využívá mozStorage, což je databázová architektura založená na SQLite, je velmi užitečným nástrojem také doplněk SQLite Manager[15], který umožňuje procházení a editaci databáze. Samozřejmostí je i vykonávání SQL příkazů[14].

2.3.4 XMPP konzole

XMPP konzole slouží k zobrazení všech příchozích a odchozích stanz. Jelikož se nepodařilo najít XMPP konzolu pro Thunderbird, jako nouzové řešení postačila XMPP konzola klienta na druhé straně, kterým byl již zmíněný Pidgin.

Kapitola 3

Analýza a návrh řešení

3.1 XMPP protokol s použitím xmpp4moz

Jako komunikační vrstva byl zvolen doplněk xmpp4moz. Samotná komunikace je díky tomu příjemně zjednodušena, ale přesto bychom měli porozumět tomu, jak komunikace protokolem XMPP funguje.

Názorně si ukážeme, jak komunikace vypadá a to jak podle volání funkcí xmpp4moz[18] tak i podle RFC, ve kterých je definována[19].

3.1.1 Stanza

Stanzy jsou základním elementem protokolu XMPP. Rozlišujeme tři druhy[7]:

- `<presence/>`: Záležitosti týkající se připojení, statusu a autorizací. Vyznačují se tím, že pokud nemají určeného příjemce, tak fungují způsobem „vysílání“ (broadcast), tedy jsou směrovány všem lidem, kteří jsou zaregistrování k odběru presencí od daného uživatele (tedy „mají autorizaci“).

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
</presence>
```

- `<message/>`: Zprávy všeho druhu. Slouží k odesílání dat způsobem „push“, tedy „odeslat a dál se nestarat“.

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Wherefore art thou, Romeo?</body>
</message>
```

- `<iq/>`: (Info/Query Požadavky, informace a odpovědi na ně) - v podstatě všechna data, která nejsou ani presence, ani message. Fungují způsobem dotaz-odpověď. Odeslání odpovědi je povinné. IQ stanzy musejí mít nastaveny typ a id.

```
<iq from='juliet@example.com/balcony' type='get' id='roster_1'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```

A odpověď je

```
<iq to='juliet@example.com/balcony' type='result' id='roster_1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          name='Romeo'
          subscription='both'>
      <group>Friends</group>
    </item>
    <item jid='mercutio@example.org'
          name='Mercutio'
          subscription='from'>
      <group>Friends</group>
    </item>
    <item jid='benvolio@example.org'
          name='Benvolio'
          subscription='both'>
      <group>Friends</group>
    </item>
  </query>
</iq>
```

3.1.2 Připojení k serveru

Pro připojení k serveru použijeme funkci `XMPP.up`

```
XMPP.up(account, postConnectionCallback);
```

Význam parametrů je následující:

- `account`: Účet, kterým se chceme připojit. Parametr může být prázdný, řetězec nebo objekt. Pokud je prázdný, xmpp4moz se dialogovým oknem dotáže na účet, který má být pro připojení použit.
- `postConnectionCallback`: Funkce, která bude zavolána po připojení. Může být prázdné.

Pokud bude `account` řetězec, očekává se ve formátu `"jméno@server/zdroj"`[1]. Pokud je parametrem objekt, očekává se ve formátu:

[1] Jako zdroj se většinou uvádí, odkud je uživatel připojen, např. doma, notebook, nebo jméno klienta. Ačkoliv XMPP podporuje více současných připojení z jednoho účtu, každý klient musí mít uveden jiný zdroj. Někteří klienti toto „jistí“ tak, že za zdroj přidají náhodnou sekvenci znaků. Tak to je i u xmpp4moz.

```
{
  jid: "jméno@server/zdroj",
  password: "*****",
  connectionHost: "server",
  connectionPort: 5223,
  connectionSecurity: 1
}
```

- `jid`: Účet, pokud není zadán, ostatní parametry jsou ignorovány a uživatel je na účet dotázán.
- `password`: Heslo, pokud není zadáno, uživatel je dotázán, ale ostatní parametry jsou akceptovány.
- `connectionHost`: Adresa serveru, na který se připojujeme. Někdy je jiná než řetězec za @ v názvu účtu.
- `connectionPort`: Port, defaultní je 5223.
- `connectionSecurity`: Šifrování SSL, defaultně je povoleno (1).

3.1.3 Zjištění stavu připojení

Stav připojení můžeme zjišťovat funkcemi

```
XMPP.isUp(account);
XMPP.isDown(account);
```

3.1.4 Vytvoření kanálu

```
XMPP.createChannel(features);
```

Návratovou hodnotou je objekt komunikační kanál. Parametrem `features` mohou být informace, které může server vyžadovat při zjišťování služeb serveru klientem. Toto my implementovat nebudeme a proto se tím nemusíme dále zabývat.

3.1.5 Registrace listeneru

Nyní, když máme komunikační kanál, vytvoříme listenery patřičně reagující na příchozí stanzy.

```
channel.on(pattern, handler);
```

Jak již napovídají názvy parametrů, vytvoří nový listener, který kontroluje, zda nově příchozí stanza odpovídá vzoru (`pattern`). Pokud ano, zavolá obslužnou funkci (`handler`). Návratovou hodnotou je samotný listener (vhodné uložit například pro pozdější zrušení listeneru). V praxi to může vypadat například takto:

```
var listener = channel.on({
    event: "message",
    direction: "in",
    stanza: function(s) {
        return s.body != unidentified;
    }
}, handleIncomingMessage);
```

Funkce `handleIncomingMessage` dostane stanzu jako parametr.

3.1.6 Odeslání stanzy

Odeslání stanzy provedeme pomocí funkce

```
XMPP.send(account, stanza);
```

3.1.7 Zrušení listeneru

Zrušení listeneru se provede voláním

```
channel.forget(listener);
```

3.1.8 Zrušení kanálu

Zrušení kanálu se provede voláním

```
channel.release();
```

3.1.9 Odpojení

Ukončení spojení se provede voláním funkce

```
XMPP.down(account);
```

3.2 Gecko

Doplněk je psán pro Mozillu Thunderbird verze 2, což byla v době psaní práce nejnovější stable verze využívající Gecko ve verzi 1.8.1. Tato verze již není aktuální a některé úkony, jako například práce s mozStorage není tak pohodlná, jak by mohla být s aktuální verzí[9].

Při implementaci doplňku nebudeme v zájmu přenositelnosti vytvářet žádné platformně závislé komponenty a podobně.

3.3 Návrh doplňku

Zde je prvotní návrh doplňku, popis gui a očekávaných funkcí.

3.3.1 Okno s kontakty

V hlavním okně bude seznam kontaktů podobný Contacts Sidebaru, narozdíl od něj bude ale zobrazovat kontakty ze všech lokálních adresářů, případně adresářů, které si uživatel zvolí (tabulka 3.1). Filtrování kontaktů bude stejné. Po připojení k IM síti a po obdržení uloženého seznamu IM kontaktů (tabulka 3.2) doplněk sloučí ty IM kontakty a kontakty z adresáře, jejichž IM adresy se budou rovnat. Pokud v seznamu IM kontaktů bude kontakt, který není v adresáři, zobrazí se v seznamu kontaktů navíc (tabulka 3.3). Kliknutím na kontakt bude mít uživatel možnost poslat IM zprávu, e-mail nebo pokud to bude možné zahájit hlasový/video hovor.

Jméno	**IM adresa**
Josef Novák	josef@novak.cz
František Dobrota	

Tabulka 3.1: Kontakty v lokálním adresáři

Stav	**Přezdívka**	**IM adresa**
Online	josef	josef@novak.cz
Pryč	marie	marie@kucerova.cz

Tabulka 3.2: Kontakty IM

Stav	**Jméno**	**IM adresa**
Online	Josef Novák	josef@novak.cz
	František Dobrota	
Pryč	marie	marie@kucerova.cz

Tabulka 3.3: Kontakty sloučené

3.3.2 Okno s rozhovorem

Otevření nového rozhovoru vytvoří novou záložku v `messagepaneboxu`[2]. Záložka bude obsahovat záznam rozhovoru, kde budou všechny přijaté a odeslané zprávy, pole pro psaní textu, tlačítka pro odeslání zprávy a nastavení písma. K dispozici budou volby jako tučné písmo, kurzíva, podtržené písmo, zvětšení nebo změnšení písma a změna barvy písma. Pokud přijde zpráva do záložky, která právě není vybrána, záhlaví záložky se dostatečně zvýrazní, aby na novou zprávu upozornilo.

[2]Box, ve kterém je po spuštění Thunderbirdu uvítací zpráva a ve kterém se po vybrání e-mailu zobrazí náhled.

3.3.3 Historie rozhovorů

Do `threadPaneBoxu`[3] přibyde nová záložka, její obsah bude strom obsahující historii rozhovorů. Vzhled bude podobný jako u seznamu e-mailů, každá položka bude jeden rozhovor. Rozhovor bychom mohli definovat různými způsoby, v tomto případě to nejspíš budou všechny zprávy pro které platí, že mezi každými dvěma po sobě jdoucími zprávami neuplynula doba delší, než například 60 minut. Pro lepší představu názorně ukázáno v tabulce 3.4. Dvojklikem na položku stromu se otevře okno se záznamem daného rozhovoru.

Rozhovor	Čas	Text
1	13:36	Ahoj, jak se máš?
1	13:45	Čau, jde to.
2	14:56	Nezajdeme na pivo?
2	15:02	Mohli bychom.

Tabulka 3.4: Zprávy - řazení do rozhovorů

3.3.4 Nastavení

Doplněk bude obsahovat základní možnosti nastavení, mezi ně určitě musí patřit uživatelské jméno, heslo a další údaje potřebné k přihlášení na IM účet. Dále zde bude na výběr, které adresáře si uživatel přeje použít pro vyhledávání kontaktů.

[3] Box, ve které je zobrazen seznam e-mailů.

Kapitola 4

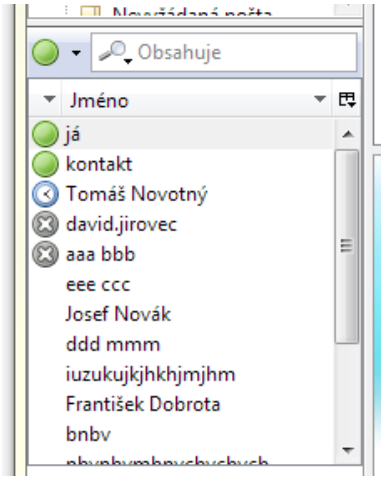
Realizace

4.1 Přehled

Vzhled seznamu kontaktů a záložky s historií je definován v souboru `overlay.xul`, jejich chování je pak definováno v samostatných souborech `js`. Chatovací záložky se v případě potřeby vytvářejí za běhu pro každý kontakt zvlášť a proto nejsou definovány v žádném `xul` souboru, vše je v `chat.js`.

4.2 Seznam kontaktů

Seznam kontaktů se zobrazuje podobně jako Contacts Sidebar pod `folderPaneBox`em[1]. Seznam je oddělen splitterem umožňujícím měnit poměr velikostí mezi seznamů složek a kontaktů.



Obrázek 4.1: Seznam kontaktů

[1] Seznam složek, v základním Thunderbirdu zabírá celý levý kraj okna.

4.2.1 Strom s kontakty

Samotný strom s kontakty je definován takto[2]:

```
<tree id="contactsTree" flex="1" sortCol="GeneratedName"
      onclick="contactsTreeClicked(event)"
      ondblclick="contactsTreeDoubleClicked(event)"
      persist="sortCol" tooltip="contactsTreeTooltip"
      onmousemove="onContactsTreeMouseMove(event)">
  <treecols id="contactsTreeCols">
    <treecol id="XMPPStatus" minwidth="20" fixed="true" class="sortDirectionIndicator"
             persist="hidden sortDirection" />
    <splitter class="tree-splitter" />
    <treecol id="GeneratedName" flex="1" label="Jméno" class="sortDirectionIndicator"
             sortDirection="ascending"
             persist="hidden width sortDirection" />
    <splitter class="tree-splitter" />
    <treecol id="PrimaryEmail" flex="1" label="E-mail" class="sortDirectionIndicator"
             persist="hidden width sortDirection" hidden="true"/>
    <splitter class="tree-splitter" />
    <treecol id="_AimScreenName" flex="1" label="IM" class="sortDirectionIndicator"
             persist="hidden width sortDirection" hidden="true"/>
    <splitter class="tree-splitter" />

    ...

  </treecols>

  <treechildren/>
</tree>
```

Je důležité, aby id sloupců odpovídaly interním názvům položek v kartě kontaktu, protože jsou využívány při vyhledávání v adresáři. Jedinou vyjímkou je sloupec `XMPPStatus`, který zobrazuje ikonku aktuálního stavu kontaktu. Ten v adresáři samozřejmě nenajdeme. Dále stojí za zmínku sloupec `_AimScreenName`, který se podle id zdá být určený pro IM síť AIM, ale Thunderbird ho v uživateli zobrazuje jako „Účet pro IM“, takže by mělo být korektní jeho použití pro náš doplněk.

Tag `<treechildren>` samozřejmě neobsahuje žádné potomky, strom budeme plnit daty za běhu podle potřeby. Zdánlivě jednoduchá možnost, jak toho docílit by mohlo být vytváření potomků `<treeitem>`, `<treerow>` a `<treecell>` za běhu, my ale zvolíme mnohem elegantnější řešení, kterým je implementace interface nsITreeView javascriptovým objektem. Toto řešení bude mít také kladný vliv na rychlost[10].

[2] Úryvek kódu zkrácen, pro názornost není třeba vypisovat všechny sloupce.

4.2.2 Implementace nsITreeView

Objekt implementující interface nsITreeView může vypadat například takto:

```
var treeView = {
  rowCount : 10000,
  getCellText : function(row,column){
    if (column.id == "namecol") return "Row "+row;
    else return "February 18";
  },
  setTree: function(treebox){ this.treebox = treebox; },
  isContainer: function(row){ return false; },
  isSeparator: function(row){ return false; },
  isSorted: function(){ return false; },
  getLevel: function(row){ return 0; },
  getImageSrc: function(row,col){ return null; },
  getRowProperties: function(row,props){},
  getCellProperties: function(row,col,props){},
  getColumnProperties: function(colid,col,props){}
};
```

Z vlastností tohoto objektu stojí za zmínku zejména:

- `rowCount` - jak název napovídá, jedná se o počet řádků stromu.
- `getCellText` - funkce, která pro každou buňku stromu vrací hodnotu, která se v ní zobrazí. Je důležité dát si pozor na to, že druhým parametrem je sloupec, nikoliv jeho id. Sloupce stromu získáme jednoduše jako `tree.columns`.
- `getImageSrc` - pro naše potřeby je tato funkce důležitá, bude se starat o zobrazení správné ikonky stavu v seznamu kontaktů.

Tento objekt nakonec stromu přiřadíme:

```
document.getElementById("id-stromu").view = treeView;
```

Výsledek vidíme na obrázku (todo přidat obrázek). Než si ukážeme, jak je toto řešení využito v našem doplňku, musíme nejdříve získat samotná data, která budeme zobrazovat.

4.2.3 Hledání v adresáři

Thunderbird umí v základu hledat pouze v jednom konkrétním adresáři. Hledání ve více adresářích najednou je funkce, po které mnoho uživatelů volá, ale dosud nebyla implementována[2].

Pro hledání se používá interface `nsIABView`. Ten bohužel není zdokumentavaný[8], musí nám proto stačit náhlédnutí do zdrojových kódů, kde najdeme:

```
NS_IMETHODIMP nsAbView::Init(const char *aURI, PRBool aSearchView,
                             nsIAbViewListener *abViewListener,
                             const PRUnichar *colID,
                             const PRUnichar *sortDirection,
                             PRUnichar **result)
```

První parametr `const char *aURI` očekává řetězec ve formátu podobném tomuto:

```
address-book?(or(first-column,c,@V)(second-col,c,@V)(third-coll,c,@V))
```

- `address-book` je URI prohledávaného adresáře. Zbytek řetězce není nutný, pokud chceme všechny položky.
- `or` je logický operátor.
- `first-column, second column...` jsou id sloupců.
- `c` je mód hledání. Dostupné možnosti jsou uvedeny v tabulce 4.1.

Parametr	Název	Česky
`c`	contains	obsahuje
`bw`	begins with	začíná
`ew`	ends with	končí
`=`	equals	je roven

Tabulka 4.1: Hledací módy abView

- `@V` je hledaný řetězec.

Zavoláním funkce `QueryInterface` s parametrem `Components.interfaces.nsITreeView` na instanci abView získáme objekt implementující interface `nsITreeView`, který už můžeme dále jednoduše použít.

Pro naše potřeby by se možná zdálo nejjednodušší pomocí abView prohledat každý adresář, z výsledného objektu pomocí funkce `getCellText` dostat kontakty, sloučit je s kontakty z IM účtu a uložit je do svého vlastního objektu implementujícího `nsITreeView`.

Problém je bohužel v tom, že toto řešení by nebylo funkční pro kontakty, které:

- Před sloučením nevyhovují vyhledávací podmínce.
- Po sloučení vyhovují vyhledávací podmínce, například díky IM adrese.

V tomto případě se sice tyto kontakty zobrazí, ale budou se zobrazovat pouze se jménem odvozeným od IM adresy a bez dalších informací.

Použijeme proto jiné, trochu těžkopádnější, ale spolehlivé a funkční řešení. Postup bude následující:

- Z každého adresáře dostaneme všechny kontakty.
- Tyto sloučíme s kontakty z IM účtu.
- Sloučené kontakty profiltrujeme pomocí regulárních výrazů.
- S pomocí takto získaných dat vytvoříme objekt implementující `nsITreeView`.

Objekt bude vypadat takto:

```
var treeView = {
  data: [],
  rowCount: 0,
  getCellText: function(row,column) {
    if (column.id != "XMPPStatus")
      return this.data[row][column.id];
    else
      return null;
  },
  getCellValue: function(row,column) {
    if (column.id == "XMPPStatus")
      return this.data[row].XMPPStatus + this.data[row].XMPPStatusText;
    else
      return null;
  },
  cycleHeader: function(column) {
    var oldDirection = column.element.getAttribute("sortDirection");
    var newDirection = (oldDirection == "ascending")
      ? "descending" : "ascending";
    column.element.setAttribute("sortDirection", newDirection);

    var tree = column.element.parentNode.parentNode;
    tree.setAttribute("sortCol", column.id);

    var currCol = tree.firstChild.firstChild;
    while (currCol) {
      if (currCol.id != column.id && currCol.localName == "treecol")
        currCol.removeAttribute("sortDirection");
      currCol = currCol.nextSibling;
    }

    var sortDir = (newDirection == "ascending") ? 1 : -1;

    function sortView(a,b) {
      if (a[column.id].toLowerCase() < b[column.id].toLowerCase()) return -1*sortDir;
      if (a[column.id].toLowerCase() == b[column.id].toLowerCase()) return 0;
      if (a[column.id].toLowerCase() > b[column.id].toLowerCase()) return 1*sortDir;
```

```
    }

    this.data.sort(sortView);
  },
  setTree: function(treebox){},
  isContainer: function(row){ return false; },
  isSeparator: function(row){ return false; },
  isSorted: function(){ return false; },
  getLevel: function(row){ return 0; },
  getImageSrc: function(row,col){
    if (col.id == "XMPPStatus")
      switch (this.data[row][col.id]) {
        case "1": return "chrome://bakalarka/skin/available.png";
        case "2": return "chrome://bakalarka/skin/away.png";
        case "3": return "chrome://bakalarka/skin/extended-away.png";
        case "4": return "chrome://bakalarka/skin/busy.png";
        case "5": return "chrome://bakalarka/skin/offline.png";
      }
    else
      return null;
  },
  getRowProperties: function(row,props){},
  getCellProperties: function(row,col,props){},
  getColumnProperties: function(colid,col,props){}
};
```

Všimněte si pole `data`, které bude obsahovat veškeré získané informace. Dále také toho, že stavy kontaktů jsou interně reprezentovány čísly od 1 do 5, ale sloupci `XMPPStatus` nevrátíme žádný text, nýbrž nastavíme odpovídající obrázek. Funkce `cycleHeader` se stará o odpovídající reakci po kliknutí na záhlaví sloupce.

Naplnění daty z adresáře získanými pomocí abView provedeme pomocí následujícího cyklu:

```
for (var i=0; i<results.rowCount; i++) {
  var newRow = {
    $card: abView.getCardFromRow(i),
    $dir: data.directoryProperties.URI,
    // status column
    XMPPStatus: "6"
  };
  for (var j in colsArray)
    newRow[colsArray[j]] = results.getCellText(i,tree.columns[colsArray[j]]);
  treeView.data.push(newRow);
}
```

Proměnné `$card` a `$dir` jsou potřebné pro možnost úpravy daného kontaktu. `XMPPStatus` mají všichni v základu nastaven hodnotu 6, která značí prázdný stav.

Spojení s kontakty z adresáře se provede tímto cyklem:

```
for (var i=0; i<treeView.rowCount; i++) {
  var jid;
  if ((jid = treeView.data[i]["_AimScreenName"]) != "")
    for each (var contact in gXMPPContacts)
      if (jid == contact.jid) {
        contact.inAB = true;
        treeView.data[i]["XMPPStatus"] = contact.status;
        break;
      }
}
```

Cyklus se pro každý kontakt z adresáře koukne, zda-li má vyplněné pole `_AimScreenName`. Pokud ano prohledá všechny kontakty v `gXMPPContacts`, což je datová struktura, ve které jsou udržovány kontakty z IM účtu. Pokud se `_AimScreenName` shoduje s `jid`, kontakt v `gXMPPContacts` je jeho proměnnou `inAB` označen a my při další práci budeme vědět, že je svázán s nějakým kontaktem z adresáře. Kontaktu v `treeView.data` přiřadíme odpovídající stav a nakonec zavoláním `break` ukončíme cyklus, protože jeden kontakt se v IM účtu nemůže vyskytovat vícekrát.

Zbylé z IM účtu, které se nespárovaly s žádným z adresáře přidáme do `treeView.data` jako další kontakty:

```
for each (var contact in gXMPPContacts) {
  if (!contact.inAB) {
   treeView.data.push({
     XMPPStatus: contact.status,
     GeneratedName: contact.name,
     _AimScreenName: contact.jid
    });
    treeView.rowCount++;
  }
}
```

Z řezězce obsahujícího vyhledávaný výraz nebo výraz připravíme regulární výrazy pro další použití:

```
function buildContactsSearchQuery(searchString, searchMode) {
  var result = [];
  var searchTerms = searchString.split(" ");

  for (var i=0; i<searchTerms.length; i++)
    if (searchTerms[i] != "")
      switch (searchMode) {
        case "0": {
```

```
          result.push(new RegExp(searchTerms[i]));
          break;
        }
        case "1": {
          result.push(new RegExp("^" + searchTerms[i]));
          break;
        }
        case "2": {
          result.push(new RegExp(searchTerms[i] + "$"));
          break;
        }
        case "3": {
          result.push(new RegExp("^" + searchTerms[i] + "$"));
          break;
        }
      }

  return result;
}
```

`searchString` je hledaný výraz, `searchMode` je číslo od 0 do 3, které značí mód vyhledávání, a to obsahuje/je na začátku/je na konci/je rovno, v tomto pořadí. Pokud výraz obsahuje více podvýrazů, které v tomto případě budou odděleny mezerou, rozdělíme ho pomocí funkce `split`. For cyklem přes všechny podvýrazy vytvoříme odpovídající regulární výrazy podle zadaného módu hledání. Podvýraz `""` ignorujeme, vznikl z několika mezer za sebou.

Následně zavoláním funkce `treeView.data.filter(filterContacts)` získáme požadovanou podmnožinu kontaktů. Funkce `filterContacts` je definována následně:

```
function filterContacts(contact) {
  for each (var exp in searchExps)
    for (var col=0; col<colsArray.length+1; col++) {
      if (col == colsArray.length) return false;
      if (contact[colsArray[col]] && contact[colsArray[col]].toLowerCase().
        search(exp) != -1) break;
    }

  return true;
}
```

Pro začátek je vhodné zdůraznit, že u daného kontakty musejí být nalezeny všechny podvýrazy. Funkce pro každý podvýraz prohledá každý kontakt, pokud nějaký podvýraz není u kontaktu nikde nalezen, funce vrací `false` a `filter` kontakt vyřadí, jinak je vráceno `true` a kontakt je ponechán. Všimněte si použití funkce `toLowerCase`, která převede řetězec na malá písmena. Parametr `searchString` je funkci `buildContactsSearchQuery` předám také po zavolání funkce `toLowerCase`, což zajistí nezávislost na zadání velkých či malých písmen.

4.2.4 Vyhledávací box

V zájmu zachování konzistence uživatelského rozhraní vychází vyhledávací box z obecného vyhledávacího boxu používaného v celém Thunderbirdu. Ten vytvoříme jednoduše definováním `textbox`u s id `searchInput`, díky XBL bindingu aplikovanému v souboru `messenger.css`. Jedná se o tento kód:

```
#searchInput {
  -moz-binding: url("chrome://messenger/content/search.xml#searchbar");
}
```

Zásadním problémem je ale fakt, že hlavní okno Thunderbirdu už jeden hledací box obsahuje, slouží k vyhledávání v e-mailech. Případné pokusy o vyřešení tohoto problému tím, že změníme id nového hledacího boxu a binding na něj aplikujeme ze svého css souboru také selžou, jelikož oba hledací boxy bodou provázané svým chováním, kvůli id vnitřních elementů. Můžeme se sice pokusit projít vnitří uzly anonymně a protože známe jejich pořadí, změnit jejich id, ale toto řešení není příliš elegantní a hlavně chování metod zůstane stejné. Nabízí se tyto dvě možnosti řešení:

- Přesunout hledací box do jiného `xul` dokumentu, který by se načetl například v `iframe` nebo `browser`. U tohoto řešení nastává problém s komunikací mezi „vnějším“ a „vnitřním“ `xul` dokumentem.
- Vzít definici původního hledacího boxu z `chrome://messenger/content/search.xml` a vytvořit kopii upravenou pro naše vlastní potřeby.

My zvolíme druhý způsob. Není třeba žádných složitých úprav, jedná se skutečně pouze o změnu několika id a parametrů funkcí. V souboru `search.xml` se nachází také binding `searchBarDropMarker`, který definuje tlačítko s obrázkem lupy, po jehož aktivaci se zobrazí menu s možnostmi hledání. Ten není třeba měnit, není zde definováno žádné konkrétní chvání, ale pouze vzhled.

Je ale třeba si ručně nadefinovat chování, které není přímo součástí bindingu, ale bylo by vhodné, aby bylo stejné jako u ostatních hledacích boxů. Patří sem věci jako:

- Zobrazování módu hledání, pokud není zadán žádný text a box nemá focus.
- Prodleva 800 ms mezi zadáním textu a hledáním. Uživatel může hledat bez prodlevy stisknutím klávesy enter.
- Reakce za změnu módu hledání.

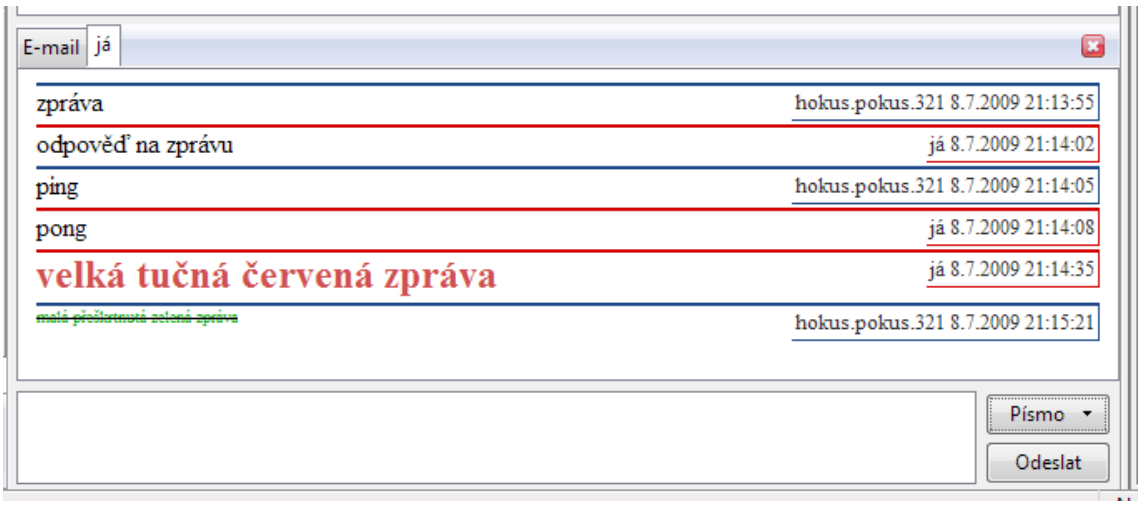
4.2.5 Tlačítko pro změnu stavu

Tlačítko je definované jako `toolbarbutton`, atribut `type` je má hodnotu `menu`, což znamená, že po kliknutí se zobrazí `menupopup`, který v našem případě obsahuje možné stavy, ze kterých si uživatel může vybrat.

Díky atributu `persist` si tlačítko v atributu `value` „pamatuje“ poslední stav a ve `value` jednotlivých `menuitem`ů si pamatuje i poslední text stavu. Kromě připojení ihned po startu Thunderbirdu se se vždy po změně stavu zeptá na nový text stavu, případně je možné ponechat minulý.

Při výpadku spojení se přepne do stavu offline a zobrazí upozornění o ztrátě spojení. Bylo by dobré také udělat automatické znovupřipojení, aktuální verze xmpp4moz bohužel obsahuje chybu, kdy pokus o připojení v době, kdy není k dispozici internet, neskončí time-outem, ale xmpp4moz stále zobrazuje připojování, to nikdy neskončí a je nutné restartovat Thunderbird.

4.3 Okno s rozhovory



Obrázek 4.2: Záložka s rozhovorem

Nové záložky s rozhovory jsou vytvářeny v místě, kde byl původně `messagepanebox`. Ten zůstal zachován, je nyní jednou ze záložek, což je ale poznat až když se nějaký rozhovor otevře, protože do té doby jsou „ouška“ záložek schovaná. Toho je docíleno nastavením atributu `collapsed` pro element `tabs`.

Pro zajímavost je možné zmínit, že pro skrytí byl původně použit atribut `hidden`, ale záložky se pak chovaly velmi nestandardně a proto je i dále v případě potřeby využíván převážně atribut `collapsed`.

Všechny elementy se zde vytvářejí a připojují na sebe za běhu javascriptem, metodami `createElement` a `appendChild`.

Záložka s rozhovory obsahuje `iframe` zobrazující odeslané a přijaté zprávy, `editor` pro zadávání textu a tlačítka pro odeslání zprávy a nastavení písma.

Atribut `id` je například pro `editor` na všech tabech z části stejný (`textInput`) a za něj je doplněno jid daného kontaktu. Jelikož jid je unikátní a pro každý jid nám stačí jedna záložka, díky čemuž můžeme použít jid jako zbytek identifikátoru.

Hlasová a video komunikace nakonec implementována nebyla, protože Google Talk plugin nemá nijak dokumentované api a zkoumat ho přímo debugováním Gmailu by bylo příliš složité.

4.3.1 Přijaté a odeslané zprávy

Jak již bylo řečeno, zprávy se zobrazují v `iframu`, což se zdá být nejjednodušší a nejlepší způsob, jelikož formátované zprávy posílané protokolem xmpp využívají html formátování. Tento `iframe` je vytvořen následujícím kódem:

```
var tabpanelElement = document.createElement("iframe");
tabpanelElement.setAttribute("id","chatLog" + contact.address);
tabpanelElement.setAttribute("flex","1");
tabpanelElement.setAttribute("src","chrome://bakalarka/skin/chat.htm");
tabpanelElement.setAttribute("value","0");
tabpanelElement.setAttribute("class","chatLog");
tabpanelElement.addEventListener("load", function() {
  this.contentDocument.body.innerHTML +=
    this.getAttribute("value").
    substring(1,this.getAttribute("value").length);
  this.setAttribute("value","1");
  this.contentDocument.body.scrollTop =
    this.contentDocument.body.scrollHeight;
}, true);
tabpanel.appendChild(tabpanelElement);
```

`iframe` po vytvoření načte stránku, která je prázdná, ale je do ní vložena css šablona, která definuje způsob zobrazení zpráv. Ty jsou pak jednoduše připojovány k contentDocument.body.innerHTML.

Problémem, který bylo třeba vyřešit je to, že první příchozí zpráva, která způsobí vytvoření záložky s rozhovorem se nemůže okamžitě zapsat, protože stránka uvnitř `iframu` není načtena okamžitě.

Toto je vyřešeno použitím atributu `value` jako bufferu. `value` zpočátku nese hodnotu `0`. Zpráva přijatá v době, kdy je `value` rovno `0` je zapsána za tuto `0`, takže `value` obsahuje `0zprava`. Po načtení stránky je zavolána obsluha event listeneru, která do `iframu` zapíše obsah `value` za prvním znakem a nastaví `value` na `1`. Další příchozí zprávy se po zjištění, že je `value` rovno `1` zapisují již přímo do `iframu`.

4.3.2 Zadávání textu

Pro zadávání textu je použit prvek `editor`, který je podobný `iframu`, ale navíc umožňuje jeho obsah měnit přímo. Je vytvořen následujícím kódem:

```
var tabpanelBoxElement = document.createElement("editor");
tabpanelBoxElement.setAttribute("id","textInput" + contact.address);
```

```
tabpanelBoxElement.setAttribute("editortype","html");
tabpanelBoxElement.setAttribute("flex","1");
tabpanelBoxElement.setAttribute("onkeypress","textInputKeyPressed(\"" +
  contact.address + "\",event);");
tabpanelBoxElement.setAttribute("class","textInput");
// enable designmode
tabpanelBoxElement.addEventListener("load", function(event) {
  this.contentDocument.designMode = "on";
  this.contentWindow.focus();
}, true);
tabpanelElement.appendChild(tabpanelBoxElement);
```

Zde není zadána žádná konkrétní stránka, použije se prázdná. Event listener zde po načtení zapne design mode, který umožňuje přímo měnit obsah uživatelem a programem vykonávat příkazy, které jako je změna písma a podobně. Následně je nastaven focus, jelikož se očekává, že se bude s tímto prvkem pracovat, tedy psát zpráva.

4.3.3 Tlačítko pro odeslání zprávy

Tlačítko má stejný efekt, jako stisknutí klávesy enter v `editoru` pro psaní textu. Jaké má odesílání zprávy fáze si probereme dále.

4.3.4 Tlačítko pro změnu formátu písma

Po kliknutí na tlačítko pro změnu formátu písma se zobrazí popup, struktura je následující:

- Tučné
- Kurzíva
- Podtržené
- Přeškrtnuté
- Velikost Písma
  - Větší
  - Normální
  - Menší
- Barva
  - Několik základních barev, netřeba vyjmenovávat...
  - Vlastní
- Vymazat formátování

Výběr nějaké možnosti ji odpovídajícím způsobem v popupu označí, což například u možnosti Tučné znamená přídání „fajfky“ (položka je typu `checkbox`, atribut `checked` bude tedy nastaven na `true`). Dále se vykoná odpovídající příkaz na `editoru` pro zadávání textu, v tomto případě to bude následující kód:

```
document.getElementById("textInput" + contact.address).
  contentDocument.execCommand("bold",false,null);
document.getElementById("textInput" + contact.address).contentWindow.focus();
```

Jeho opětovné zavolání by tučné písmo zase zrušilo. Dále si všimněme, že se nastaví focus `editoru`, takže můžeme rovnou psát.

Možnost Vlastní barva vybídne uživatele k zadání html kódu požadované barvy.

4.3.5 Obsluha příchozí zprávy

Příchozí zprávu zachytí tento listener:

```
channel.on({
  event: 'message',
  direction: 'in',
  stanza: function(s) {
    return s.body != undefined;
  }
}, incomingMessage);
```

Funkce `incomingMessage` provede následující:

- Pokud pro kontakt s daným JID není vytvořena záložka s rozhovorem, vytvoří se.
- Do `iframu` je zapsána nová zpráva, html kód je v následujícím formátu:

  ```
  <div class="chat-message-in">
    <span class="chat-header-in">
      <span class="chat-header-name">jméno</span>
      <span class="chat-header-time">datum_a_čas</span>
    </span>
    text_zprávy
  </div>
  ```

  Než vložíme text zprávy, je třeba zjistit, zda-li zpráva obsahuje formátování. Provedeme to takto:

  ```
  if (mess.stanza.ns_xhtml_im::html != undefined)
    toWrite += mess.stanza.ns_xhtml_im::html;
  else
    toWrite += mess.stanza.body.toString().replace(/\n/g,"<br/>");
  ```

`replace(/\n/g,"<br/>")` u neformátované zprávy nahradí všechny nové řádky tagem `<br/>`. Toto je třeba provést, protože i tato neformátovaná zpráva bude jako html vložena do `iframu` a `\n` by se neprojevily.

- Posun `iframu` dolů tak, aby byla vidět poslední zpráva. To se provede takto:

```
chatLog.contentDocument.body.scrollTop =
  chatLog.contentDocument.body.scrollHeight;
```

- Uložení zprávy do databáze tímto voláním:

```
gDB.executeSimpleSQL('INSERT INTO "history"
  ("jid", "direction", "date", "message")
  VALUES ("' + from + '", "in", "' + now.toString() + '", "
  mess.stanza.body.toString().replace(/\"/g,"\"\"") + '")');
```

`replace(/\"/g,"\"\"")` nahradí všechny výskyty jedné uvozovky za dvě uvozovky za sebou. Dvě uvozovky se považují za jednu uvozovku, která je ale obsah řetězce. Kdyby zůstala jedna samotná uvozovka, byla by považována za začátek nebo konec řetězce.

- Aktualizace stromu s historií.

- Upozornění uživatele na novou zprávu. To se děje následujícími:

  – Zvýrazněním popisu odpovídající záložky. Toto se děje pouze pokud nebyla aktivní.

  – Zobrazením upozorňujícího okénka s částí textu zprávy. Toto provádí následující kód:

```
var alertsService = Components.classes["@mozilla.org/alerts-service;1"].
                      getService(Components.interfaces.nsIAlertsService);

alertsService.showAlertNotification("chrome://bakalarka/skin/xmpp.png",
  name, text.toString().length > gPrefs.getIntPref("xmpp.cropAlerts")
  ? text.substring(0,gPrefs.getIntPref("xmpp.cropAlerts")) + "..."
  : text, false, "", null);
```

  Jak je z kódu vidět, v `about:config` je možné klíčem `xmpp.cropAlerts` nastavit, kolik znaků ze zprávy se v upozornění zobrazí.

  – Zavoláním funkce `window.getAttention`, což jednoduše způsobí zablikání okna a jeho ikony na taskbaru, pokud okno nemělo focus.

4.3.6 Obsluha odchozí zprávy

Obsluha odchozí zprávy samozřejmě žádný listener registrovaný na `channelu` nepotřebuje, řeší se přímo. Postup je následující:

- Jelikož bude kromě html formátované ve stanze odeslána i její čistě textová verze je nutné ji nějak získat. Provedeme to rekurzivním průchodem DOM stromem dokumentu v `editoru`. Metoda pro tento účel vypadá takto:

```
var bodyNoHtml = "";

function noHtml(root) {
  if (root.nodeName == "#text")
    bodyNoHtml += root.nodeValue;
  else if (root.childNodes)
    for (var i=0; i<root.childNodes.length; i++)
      noHtml(root.childNodes[i]);
}
```

- Odeslání stanzy:

```
XMPP.send(account,
'<message to="' + to + '">
   <body>' + bodyNoHtml + '</body>
   <html xmlns="http://jabber.org/protocol/xhtml-im">
     <body xmlns="http://www.w3.org/1999/xhtml">' +
       textInput.contentDocument.body.innerHTML.
         replace(/<br>/g,"<br/>") + '
     </body>
   </html>
 </message>');
```

`replace(/<br>/g,"<br/>")` nahrazuje všechny neuzavřené tagy `<br>`, které vkládá `editor` za uzavřené `<br/>`.

- Zápis zprávy do `iframu`. Ten probíhá analogicky jako u příchozí zprávy, pouze třídy chat-message-in a chat-header-in jsou nahrazeny chat-message-out a chat-header-out.
- Posun `iframu` dolů na poslední zprávu, opět stejný jako u příchozí zprávy.
- Uložení zprávy do databáze. Analogické s příchozí zprávou.
- Obnovení stromu s historií.
- Vymazání `editoru`. Provede se takto:

```
textInput.contentDocument.body.innerHTML = "";
textInput.contentDocument.execCommand("delete",false,null);
```

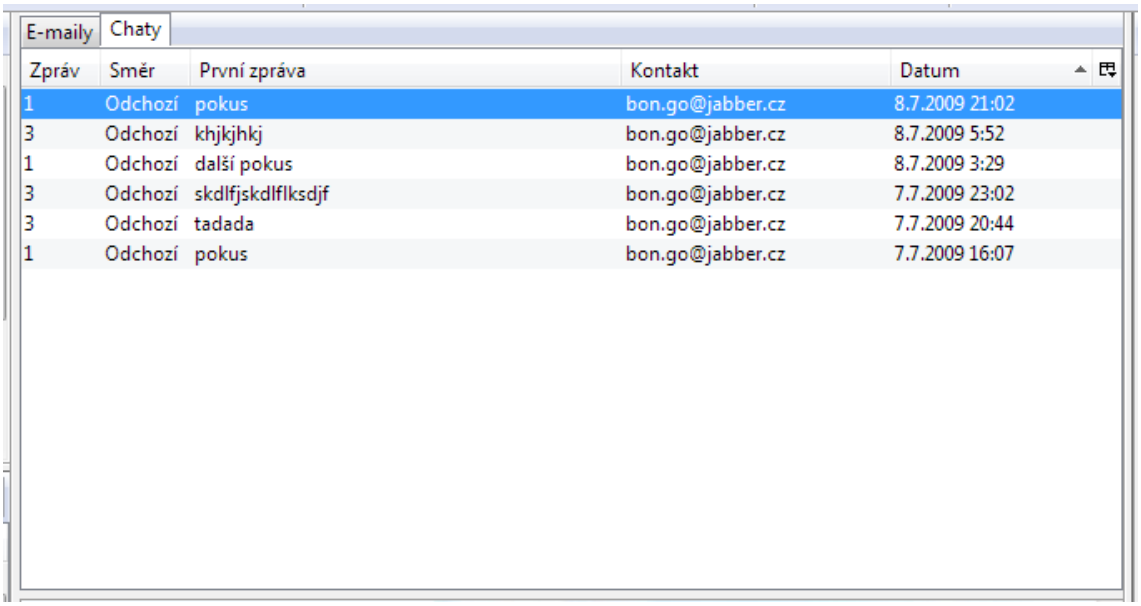
`delete` je provedeno proto, že po přiřazení prázdného řetězce do `body.innerHTML` zmizí v `editoru` kurzor. Po vykonání `delete` se opět objeví, a protože je v tu chvíli `editor` stejně prázdný, nic jiného se nestane.

- Znovunastavení formátu písma, jelikož byl vymazáním `editoru` zrušen.
- Nastavení focusu na `editor`.

4.4 Historie rozhovorů

V `threadPaneBoxu` byly vytvořeny záložky, do jedné byl přesunut `threadTree` a v další byl vytvořen `historyTree`, který obsahuje jednotlivé uložené rozhovory. Sloupce jsou:

- Počet zpráv.
- Směr. Udává, zda první zpráva rozhovoru byla odchozí či příchozí, čili která strana rozhovor začala.
- První zpráva.
- Kontakt. JID druhé strany.
- Datum.



Obrázek 4.3: Historie

4.4.1 Způsob uložení historie

Data jsou uložena v mozStorage, což je SQLite databáze poskytovaná Geckem. Veškerá historie je uložena v jedné tabulce, která obsahuje sloupce `jid`, `direction`, `date` a `message`. Všechny sloupce jsou typu `TEXT`, i `date`, které je uloženo ve formátu, který vrací javascript po zavolání metody `toString` na objekt třídy `Date`. Databáze je uložena v souboru xmpp-history.sqlite v profilu uživatele.

4.4.2 Načtení historie

Nejdříve se načtou z databáze všechny kontakty, pro které je uložena nějaká zpráva:

```
var statement =
  Components.classes["@mozilla.org/storage/statement-wrapper;1"].
  createInstance(Components.interfaces.mozIStorageStatementWrapper);
statement.initialize(gDB.
  createStatement('SELECT DISTINCT "jid" FROM "history"'));

try {
  while (statement.step()) contacts.push(statement.row.jid);
} finally {
  statement.reset();
}
```

Dále je třeba pro každý kontakt načíst zprávy a rozdělit je do rozhovorů. Rozhovor trvá od odeslání první zprávy do doby, kdy časový rozdíl mezi právě zpracovávanou zprávou a minulou zpracovávanou zprávou nepřekročí určenou hodnotu. Tato hodnota je uložena v `about:config` pod klíčem `xmpp.delayBetweenChats` a základní hodnota je 3600 sekund, čili jedna hodina. Rozdělení se provede následujícím kódem:

```
for each (var contact in contacts) {
  statement =
    Components.classes["@mozilla.org/storage/statement-wrapper;1"].
    createInstance(Components.interfaces.mozIStorageStatementWrapper);
  statement.initialize(gDB.
    createStatement('SELECT rowid, direction, date, message ' +
    'FROM "history" WHERE "jid" = "' + contact + '"'));

  try {
    var delay = gPrefs.getIntPref("xmpp.delayBetweenChats");
    var oldDate = new Date(0);

    while (statement.step()) {
      var newDate = new Date(statement.row.date);

      if (newDate - oldDate > 1000 * delay) {
```

```
        data.push({
          historyTreeContact: contact,
          historyTreeMessage: statement.row.message,
          historyTreeMessageCount: 0,
          date: newDate.valueOf(),
          rowID: statement.row.rowid
        });

        data[data.length-1].historyTreeDirection =
          (statement.row.direction == "in")
          ? "Příchozí" : "Odchozí";

        // minutes should begin with zero if they are smaller then 10
        var min = (newDate.getMinutes() < 10) ? "0" + newDate.getMinutes()
          : newDate.getMinutes();

        data[data.length-1].historyTreeTime = newDate.getDate() + "." +
          ((newDate.getMonth())+1) + "." + newDate.getFullYear() + " " +
          newDate.getHours() + ":" + min;
      }

      data[data.length-1].historyTreeMessageCount++;
      oldDate = newDate;
    }
  } finally {
    statement.reset();
  }
}
```

Takto získaná data jsou nakonec použita pro vytvoření objektu implementujícího interface `nsITreeView` podobně jako v případě stromu s kontakty, takže není třeba tento proces znovu detailně popisovat.

4.4.3 Zobrazení rozhovoru z historie

Rozhovory načtené z historie se zobrazují v novém okně. Toto okno obsahuje `iframe`, data do něj jsou načtena podobným způsobem jako při rozdělování celé historie do jednotlivých rozhovorů. Počátek otevíraného rozhovoru se získá jako `rowid` první zprávy, které je pro každý rozhovor uloženo v `historyTree` a získá funkcí `getCellValue` na sloupec `historyTreeMessageCount` v označeném řádku.

Tento `iframe` opět načítá html dokument obsahující pouze vloženou css šablonu, podle které jsou formátovány vložené zprávy. Výpis probíhá takto:

```
log.contentDocument.body.innerHTML +=
  '<div class="chat-message-' + dir + '">' +
```

```
  '  <span class="chat-header-' + dir + '">' +
  '    <span class="chat-header-name">' + name + '</span>' +
  '    <span class="chat-header-time">' + dateTime + '</span>' +
  '  </span>' +
     statement.row.message +
  '</div>';
```

4.4.4 Mazání z historie

Historii je možné také mazat, a to po celých rozhovorech. Jelikož známe `rowid` první zprávy, je třeba získat ještě `rowid` poslední zprávy. To provedeme takto:

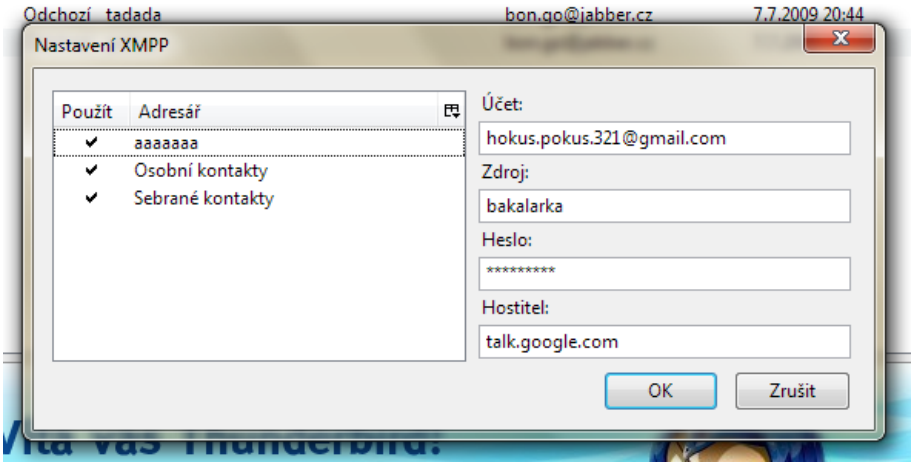
```
try {
  var delay = gPrefs.getIntPref("xmpp.delayBetweenChats");
  var oldDate = new Date(0);
  var end;

  while (statement.step()) {
    var newDate = new Date(statement.row.date);

    if (oldDate.valueOf() != 0 && newDate - oldDate > delay * 1000) break;

    end = statement.row.rowid;

    oldDate = newDate;
  }
} finally {
    statement.reset();
}
```

Nyní máme v proměnné `end` `rowid` poslední zprávy rozhovoru a celý ho můžeme vymazat, což provedeme takto:

```
statement.initialize(gDB.
  createStatement('DELETE FROM "history" WHERE "jid" = "' + contact +
  '" AND rowid >= ' + start + ' AND rowid <= ' + end));
```

4.5 Nastavení

Dialog s nastavením zobrazuje strom s checkboxy pro výběr adresářů, které budou v seznamu kontaktů využívány a nastavení účtu XMPP.



Obrázek 4.4: Nastavení

4.5.1 Výběr adresářů

Strom s checkboxy není úplně typický, proto se blíže podíváme na způsob jeho implementace.

```
<tree id="XMPPSettingsAbTree" flex="1" editable="true">
  <treecols>
    <treecol id="XMPPSettingsAbTreeUse" label="Použít" type="checkbox"
             editable="true"/>
    <treecol id="XMPPSettingsAbTreeName" label="Adresář" width="200"/>
  </treecols>

  <treechildren/>
</tree>
```

Všimněme si, co je zde jinak než obvykle:

- `tree` má atribut `editable` s hodnotou `true`.
- `treecol` má atribut `type` nastaven na `checkbox` a atribut `editable` na `true`.

Toto samo o sobě ale nestačí, jelikož by nebyly vidět „fajfky“. Ty je třeba definovat css stylem, například takto:

```
treechildren::-moz-tree-checkbox {
  /* unchecked checkbox treecells */
  list-style-image: none;
}

treechildren::-moz-tree-checkbox(checked) {
```

```
  /* checked checkbox treecells*/
  list-style-image: url("chrome://global/skin/checkbox/cbox-check.gif");
}

treechildren::-moz-tree-checkbox(disabled) {
  /* disabled checkbox treecells */
  list-style-image: url("chrome://global/skin/checkbox/cbox-check-dis.gif");
}
```

Vidíme, že obrázky jsou obsaženy přímo v Thunderbirdu, skutečně jde pouze o to nastavit, že se mají používat.

4.5.2 Nastavení účtu

Zadávají se tyto údaje:

- Účet
- Zdroj
- Heslo
- Hostitel

Hostitel, čili adresa serveru, ke kterému se budeme připojovat se vyplní automaticky, a to tak, že bude:

- talk.google.com pro účty končící na @gmail.com a @googlemail.com.
- Řetězec za znakem @ pro ostatní.

Tohoto chování je dosaženo následujícím kódem:

```
if ((account.search("@gmail.com$") != -1) ||
    (account.search("@googlemail.com$") != -1))
  hostTextbox.value = "talk.google.com";
else
  hostTextbox.value = account.replace(/.*@/,"");
```

Kapitola 5

Testování

5.1 Komunikace s jinými klienty

Doplněk byl testován při komunikaci s klienty Pidgin, QIP Infium a s jinou instancí sama sebe.

- Pidgin: Jako jediný problém se ukázalo částečně nekompatibilní formátování. Doplněk přijal od Pidginu všechno bez prolblému, ale nazpět nefungovalo vše. Na vině je patrně fakt, že v některých případech dosahují Pidgin a doplněk stejného výsledku jinak. Pro vysvětlení se podívejme na tyto zachycené stanzy obsaující text psaný kurzívnou textu.

  Odesláno Pidginem:

```
<message type='chat' id='purpleeed325e0'
         to='hokus.pokus.321@gmail.com/bakalarka5884FF62'>
  <x xmlns='jabber:x:event'>
    <composing/>
  </x>
  <body>pokus</body>
  <html xmlns='http://jabber.org/protocol/xhtml-im'>
    <body xmlns='http://www.w3.org/1999/xhtml'>
      <em>pokus</em>
    </body>
  </html>
</message>
```

  Odesláno Thunderbirdem:

```
<message from='hokus.pokus.321@gmail.com/bakalarka5884FF62'
         to='bon.go@jabber.cz' id='_12470251222211007'>
  <body>pokus</body>
  <html xmlns='http://jabber.org/protocol/xhtml-im'>
    <body xmlns='http://www.w3.org/1999/xhtml'>
```

```
      <span style='font-style: italic;'>pokus</span>
    </body>
  </html>
  <x xmlns='google:nosave' value='disabled'/>
  <record xmlns='http://jabber.org/protocol/archive' otr='false'/>
</message>
```

Stanza odeslaná naším doplňkem ale odpovídá doporučením pro xhtml formátované zprávy[17].

- QIP Infium: Tento klient byl zvolen jako zástupce klientů nepoužívajících formátování textu. Veškerá komunikace byla bezproblémová.
- Jiná instance sama sebe: Komunikace byla bezproblémová a veškeré formátování bylo podle očekávání akceptováno.

5.2 Srovnání s existujícími řešeními

Pro blížší srovnání je vhodný snad jenom Sameplace. Zde jsou výhody našeho doplňku:

- Je použitelný nejen jako IM klient, ale i pro zobrazení kontaktů v postranním panelu přímo v hlavním okně Thunderbirdu s možností hledání v libovolném počtu adresářů najednou.
- Je postaven na mnohem decentnější grafice, která se nesnaží lišit od zbytku systému.
- Ukládá historii intuitivním způsobem do jednotlivých rozhovorů.

Mezi výhody Sameplacu patří:

- Pokročilejší funkce jako správa rosteru, načítání avatarů a podobně.
- Náš doplněk může být při práci s většími adresáři a/nebo větším rosterem poněkud pomalý, jelikož má svůj specifický způsob práce se seznamem kontaktů, který vniká spojením adresáře Thunderbirdu a rosteru XMPP účtu.

Kapitola 6

Závěr

Cíle práce byly splněny, autor práce, frameworku Gecko původně neznalý, se s ním naučil pracovat. Výsledkem je funkční doplněk sloužící primárně jako IM klient využívající protokol XMPP.

Další pokračování práce je možné, a to buď ve směru optimalizace stávajícího doplňku a přidávání podpory pro další funkce nabízené protokolem XMPP, nebo rozšíření doplňku o funkce, které uživatelům zjednoduší a zpříjemní používání poštovního klientu Mozilla Thunderbird.

Literatura

[1] Doplňky aplikace Thunderbird.
https://addons.mozilla.org/cs/thunderbird/, stav z 8. 7. 2009.

[2] Bug 170270 - Enable search in multiple/all address books.
https://bugzilla.mozilla.org/show_bug.cgi?id=170270, stav z 8. 7. 2009.

[3] Contacts Sidebar - Thunderbird extension.
http://jpeters.no-ip.com/extensions/?page=tb_cs, stav z 8. 7. 2009.

[4] Doplněk DOM Inspector :: Doplňky aplikace Thunderbird.
https://addons.mozilla.org/cs/thunderbird/addon/1806, stav z 8. 7. 2009.

[5] Gmail.
http://gmail.com, stav z 8. 7. 2009.

[6] Instantbird.
http://instantbird.com/, stav z 8. 7. 2009.

[7] Stanza – Jabber.cz Wiki.
http://www.jabber.cz/wiki/Stanza, stav z 8. 7. 2009.

[8] Interfaces - MDC.
https://developer.mozilla.org/en/Interfaces, stav z 8. 7. 2009.

[9] Storage - MDC.
https://developer.mozilla.org/en/Storage, stav z 8. 7. 2009.

[10] nsiTreeView - MDC.
https://developer.mozilla.org/en/nsITreeView, stav z 8. 7. 2009.

[11] Pidgin, the universal chat client.
http://www.pidgin.im/, stav z 8. 7. 2009.

[12] SamePlace | Instant Messenger for Mozilla Firefox and Thunderbird.
http://www.sameplace.cc/, stav z 8. 7. 2009.

[13] Spicebird | Open Source Collaboration.
http://www.spicebird.com/, stav z 8. 7. 2009.

[14] SQLite Documentation.
http://www.sqlite.org/docs.html, stav z 8. 7. 2009.

[15] SQLite Manager :: Add-ons for Thunderbird.
`https://addons.mozilla.org/en-US/thunderbird/addon/5817`, stav z 8. 7. 2009.

[16] Venkman Javascript Debugger project page.
`http://www.mozilla.org/projects/venkman/`, stav z 8. 7. 2009.

[17] XEP-0071: XHTML-IM.
`http://xmpp.org/extensions/xep-0071.html`, stav z 8. 7. 2009.

[18] Api Reference.
`http://wiki.github.com/bard/sameplace/api-reference`, stav z 8. 7. 2009.

[19] XMPP RFCs.
`http://xmpp.org/rfcs/`, stav z 8. 7. 2009.

Kapitola 7

Seznam použitých zkratek

DOM Document Object Model

GPL General Public License

IM Instant messaging

JID Jabber ID

LGPL Lesser General Public License

MPL Mozilla Public License

XMPP Extensible Messaging and Presence Protocol

XUL XML User Interface Language

Kapitola 8

Obsah přiloženého CD

install/ Instalace Mozilly Thunderbird, doplňků.

source/ Zdrojové kódy projektu.

text/ Text bakalářské práce v elektronické podobě.

text/source/ Zdrojové kódy textu bakalářské práce.

readme.txt Postup instalace.