

Semesterarbeit

“Integrating Spamato into Thunderbird”

Betreuender Assistent: Keno Albrecht
Betreuender Professor: Prof. Dr. R. Wattenhofer

Sommersemester 2006

Markus Neidhart,
ETH Zürich

18. April 2006

Inhaltsverzeichnis

1	Einführung	3
1.1	Ziel der Arbeit	3
1.2	Vorgehen in diesem Dokument	3
1.3	Zusätzliche Dateien	4
2	Implementation	5
2.1	Download von <i>Thunderbird</i>	5
2.2	Kompilieren von <i>Thunderbird</i>	5
2.3	Installieren von <i>Thunderbird</i>	6
2.4	Änderungen in <i>Thunderbird</i>	7
2.4.1	Anlegen des <i>Spamato</i> -Verzeichnisses	7
2.4.2	spamato.js zu “chrome/messenger” hinzufügen	7
2.4.3	Änderungen in mailWindowOverlay.xul	8
2.4.4	Änderungen in primaryToolbar.css	9
2.4.5	Neue Icons in classic.jar einfügen	10
2.4.6	Änderungen in messenger.dtd	10
2.4.7	Änderungen in msgMail3PaneWindow.js	11
2.4.8	Änderungen in mailWindowOverlay.js	12
2.4.9	Änderungen in mailCommands.js	12
2.4.10	Fazit	13
3	Aufgetauchte Probleme während der Arbeit	15
3.1	Behandlung von <i>IMAP</i> -Mails	15
3.2	<i>Thunderbird</i> -eigener Junkfilter	16
3.3	Dokumentation von <i>Thunderbird</i>	16

1 Einführung

1.1 Ziel der Arbeit

Ziel meiner Arbeit war es, den in *Java* geschriebenen Spamfilter *Spamato* direkt in den *Thunderbird*-Quellcode zu integrieren, anstatt ihn wie bisher als Extension in *Thunderbird* zu installieren. Dabei soll der in *Thunderbird* bereits vorhandene Spammechanismus - also sowohl die visuelle Markierung als Spam als auch das Verhalten von *Thunderbird* auf Spammessages - möglichst genau übernommen werden. Der Unterschied jedoch soll sein, dass zur Erkennung und Bestimmung von Spammessages nicht der *Thunderbird*-eigene Spamfilter zum Einsatz kommt, sondern eben *Spamato*.

1.2 Vorgehen in diesem Dokument

In diesem Bericht über meine Arbeit werde ich Schritt für Schritt erklären, wie man von den *Thunderbird*-Quellen im Internet zu einer lauffähigen Version auf dem lokalen PC kommt. Die Änderungen kann man im Wesentlichen in zwei Teilen zusammenfassen:

1. Vorbereiten von *Thunderbird*:

Dieser Teil beinhaltet erstens den Download der *Thunderbird*-Quellen, zweitens das Kompilieren dieser Quellen und drittens die Installation von *Thunderbird*. Das Vorgehen in diesem Teil ist dabei abhängig vom jeweiligen Betriebssystem. Ich beschreibe das Vorgehen anhand des Betriebssystems *Linux*. Die *Linux*-Distribution, die ich dabei verwende, ist *Ubuntu* 5.10 (<http://www.ubuntu.com>).

Wird in diesem Zusammenhang ein anderes Betriebssystem verwendet, so empfehle ich, die drei Punkte (Download, Kompilieren und Installation von *Thunderbird*) nach folgender Anleitung für das jeweils verwendete Betriebssystem durchzuführen: http://developer.mozilla.org/en/docs/Build_Documentation.

Schlussendlich sollte man eine lauffähige Version von *Thunderbird* im "Originalzustand" in einem selbst gewählten Verzeichnis liegen haben.

Man kann natürlich auch eine fertig kompilierte Version installieren, und diese nachher anpassen.

2. Einbinden von *Spamato*:

In diesem Teil beschreibe ich, wie diese Version von *Thunderbird* nun so geändert wird, damit als Spamfilter *Spamato* zum Einsatz kommt. Dieser Teil ist dabei grundsätzlich unabhängig vom Betriebssystem. Es ist aber möglich, dass für den Umgang mit ".jar"-Dateien (abhängig vom Betriebssystem) andere Programme verwendet werden.

1.3 Zusätzliche Dateien

Um alle untenstehenden Änderungen nachvollziehen zu können, sind folgende zusätzliche Dateien notwendig, die ich in meiner Arbeit mitliefere:

1. *Spamato*-Dateien:

Diese Dateien stellen den *Spamato* Spamfilter dar, welcher später innerhalb von *Thunderbird* als externer *Java*-Prozess gestartet wird. Sie beinhalten drei Verzeichnisse (“jars”, “plugins” und ein leeres Verzeichnis “spamato”).

2. Die Datei “spamato.js”:

In “spamato.js” sind sämtliche Funktionen, Variablen und Konstanten definiert, die zum Starten und Kommunizieren von *Thunderbird* mit der *Java*-Applikation *Spamato* (bzw. dem “Localserver”) wichtig sind.

2 Implementation

2.1 Download von Thunderbird

Als erstes legt man sich ein Verzeichnis an, in das man *Thunderbird* downloaden wird. Dort kompiliert man später auch *Thunderbird*. In meinem Fall ist dies ein Verzeichnis “tb.compile”, das ich in meinem persönlichen Ordner erstelle (/home/myName/tb.compile/). Danach öffnet man ein neues *Terminal* und wechselt in dieses Verzeichnis.

Nun muss man sich die Quelldateien von *Thunderbird* aus dem Internet laden. Dies geht am einfachsten via *CVS* (Concurrent Versioning System). Grundsätzlich folge ich der *Mozilla*-Dokumentation (http://developer.mozilla.org/en/docs/Mozilla_Source_Code_Via_CVS):

```
cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot login
```

Dies ist der Login ins *CVS*-repository, gefolgt von der Passworteingabe (anonymous).

```
cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot co mozilla/client.mk
```

Damit lädt man sich die Datei “client.mk” in ein neues Verzeichnis “mozilla” herunter. Genau in dieses Verzeichnis wechselt man nun:

```
cd mozilla
```

Nun macht man einen Checkout des Projektes:

```
make -f client.mk checkout MOZ_CO_PROJECT=mail
```

Die Angabe von “mail” hinter `MOZ_CO_PROJECT` bedeutet, dass nur das *Thunderbird*-Projekt heruntergeladen wird.

2.2 Kompilieren von Thunderbird

Jetzt wird im “mozilla” Verzeichnis eine Textdatei namens “.mozconfig” angelegt, die folgenden Inhalt enthält:

```
. $topsourcedir/mail/config/mozconfig

mk_add_options MOZ_CO_PROJECT=mail
ac_add_options --enable-optimize
ac_add_options --disable-debug
ac_add_options --enable-default-toolkit=gtk2
ac_add_options --enable-xft
```

```
ac_add_options --disable-static --enable-shared

# jar packaging(jar|flat|symlink), default is jar
#ac_add_options --enable-chrome-format=flat
```

Der letzte Punkt (`ac_add_options --enable-chrome-format=flat`, hier auskommentiert) bestimmt, wie die Dateien im “Chrome”-Verzeichnis (siehe weiter unten bei “Änderungen in *Thunderbird*”) verwaltet werden.

Im Normalfall sind diese Dateien in sogenannten jar-Packages abgelegt (z.B. “messenger.jar”). Deshalb muss eine zu ändernde Datei zuerst aus dem Package entpackt, dann geändert und wieder ins Package abgelegt werden. Ich werde dieses Vorgehen weiter unten an einem Beispiel für die Datei “mailWindowOverlay.xul” beschreiben.

Wird “flat” ausgewählt, dann sind Dateien nicht in einem jar-Package, sondern normal in einem Verzeichnis (also z.B. Verzeichnis “messenger”).

Die Variante “symlink” arbeitet scheinbar mit symbolischen Links auf die Quellen, was aber in diesem Fall nicht zu empfehlen ist, da wir eine *Thunderbird*-Version wollen, die unabhängig von den Quellen ist.

Je nach System jedoch kann die “.mozconfig” Datei variieren. Eine gute Übersicht ist unter http://developer.mozilla.org/en/docs/Configuring_Build_Options zu finden.

Danach kann *Thunderbird* kompiliert werden. Dies geschieht mit dem folgenden Kommando:

```
make -f client.mk build
```

2.3 Installieren von Thunderbird

Nun wird *Thunderbird* installiert, d.h. es wird ein komprimiertes Archiv erstellt, das die ganze Applikation enthält. Dazu den folgenden Befehl im “mozilla”-Verzeichnis aufrufen:

```
make -C mail/installer
```

Dieses Archiv findet man nun im Unterverzeichnis “dist” (`thunderbird-*.linux-i686.tar.bz2`). Dieses kann jetzt an einen beliebigen Ort entpackt werden (`bunzip2/tar`). In meinem Fall tue ich folgendes:

```
cp dist/thunderbird-*.linux-i686.tar.bz2 /home/myName/
```

```
cd /home/myName/
```

```
bunzip2 thunderbird-*.linux-i686.tar.bz2
```

```
tar xvf thunderbird-*.linux-i686.tar
```

Jetzt findet man unter `/home/myName/thunderbird/` (im Folgenden als *Thunderbird*-Ordner bezeichnet) die *Thunderbird*-Version, die als nächstes geändert wird.

2.4 Änderungen in Thunderbird

Jetzt werden die Änderungen realisiert, damit *Thunderbird* als Einheit mit *Spamato* gestartet werden kann. Dabei sind sämtliche relevanten Dateien im sogenannten “Chrome”-Verzeichnis zu finden. Das “Chrome”-Verzeichnis (Unterverzeichnis des *Thunderbird*-Ordner) enthält die Gesamtheit aller Dateien, die das *User Interface* von *Thunderbird* ausmachen und ist in 3 Teile unterteilt:

1. Content:

Unter “Content” finden sich Dateien, die das Verhalten von *Thunderbird* beeinflussen und haben meist die Endungen “.xul” oder “.js”.

“.xul”-Dateien definieren den Aufbau von Fenstern, Menüs sowie Buttons, usw. In “.xul”-Dateien werden auch jene “.js”-Dateien angegeben, welche mit dem Laden des entsprechenden “.xul”-Files auch gerade geladen werden müssen. Ausserdem kann auch direkt auf *JavaScript*-Funktionen verwiesen werden (z.B. was passiert, wenn ein Button angeklickt wird).

In den “.js”-Dateien wird die ganze Programmlogik von *Thunderbird* beschrieben.

2. Locale:

Unter “Locale” befinden sich Dateien, welche für den Multi-Language-Support von *Thunderbird* zuständig sind und haben die Endungen “.dtd” und “.js” (Preferences).

“.dtd”-Dateien (*Document Type Definition*) speichern dabei sprachspezifische Informationen für Labels und Tooltips, welche direkt aus “.xul”-Dateien heraus referenziert werden.

3. Skin:

Unter “Skin” befinden sich Dateien, welche das Erscheinungsbild von *Thunderbird* beeinflussen und haben meist die Endung “.css”.

“.css”-Dateien (*Cascading Stylesheets*) beschreiben das Aussehen bestimmter Elemente, welche in “.xul”-Dateien verwendet werden. Die Beschreibung umfasst zum Beispiel Schriftart, Schriftgrösse oder auch Informationen zu Icons.

Ausserdem befinden sich auch Bilddateien (Icons) unter “Skin”.

2.4.1 Anlegen des Spamato-Verzeichnisses

Zu Beginn müssen die *Spamato*-Dateien in ein Verzeichnis gelegt werden. In meinem Beispiel lege ich unter dem *Thunderbird*-Ordner ein Verzeichnis “spamato” an und speichere dort die oben genannten drei Verzeichnisse der *Spamato*-Dateien.

2.4.2 spamato.js zu “chrome/messenger” hinzufügen

In “spamato.js” muss nun der Ort des *Spamato*-Verzeichnisses angegeben werden. Dazu muss die folgende Konstante entsprechend angepasst werden:

```
const SPAMATO_DIR = "...";
```

In meinem Beispiel wäre dies `"/home/myName/thunderbird/spamato"`. Zu Beachten ist hier, dass unter Windows Backslashes doppelt angegeben werden müssen:

```
C:\\Programme\\Mozilla\\...
```

Nun muss diese angepasste Datei `"spamato.js"` ins Package `"messenger.jar"` - und dort nach `"content/messenger"` - kopiert werden.

2.4.3 Änderungen in `mailWindowOverlay.xul`

Mit einem Archiv-Manager (ich verwende in meinem Fall z.B. den *File Roller* von *Gnome*) die Datei `"messenger.jar"` öffnen. Im Archiv-Manager nun ins Unterverzeichnis `"content/messenger/"` wechseln und die Datei `"mailWindowOverlay.xul"` an einen temporären Ort entpacken (z.B. Drag&Drop auf den Desktop) und zum Bearbeiten öffnen (Texteditor). Danach folgende Änderungen vornehmen:

- Datei `"spamato.js"` einbinden. Dies geschieht, indem nach der Zeile

```
<script type="application/x-javascript"
  src="chrome://global/content/viewZoomOverlay.js"/>
```

folgendes hinzugefügt wird:

```
<!-- next line loads spamato.js -->
<script type="application/x-javascript"
  src="chrome://messenger/content/spamato.js"/>
```

- Der `"Junk"/"Not Junk"`-Button wird angepasst, und zwar so, dass `"Junk"` und `"Not Junk"` zur gleichen Zeit sichtbar sind. Ausserdem wird der `"Statistik"`-Button von *Spamato* hinzugefügt. Dazu den Code

```
<toolbaritem id="button-junk">
  <deck id="junk-deck" observes="button_junk">
    <toolbarbutton class="toolbarbutton-1 junk-button"
      label="%junkButton.label;"
      tooltip="%junkButton.tooltip;"
      observes="button_junk"
      oncommand="goDoCommand('button_junk')"/>
    <toolbarbutton class="toolbarbutton-1 junk-button"
      label="%notJunkButton.label;"
      tooltip="%notJunkButton.tooltip;"
      observes="button_junk"
      oncommand="goDoCommand('button_junk')"/>
  </deck>
</toolbaritem>
```

durch die folgende Änderung ersetzen:

```
<!-- now comes the SPAMATO code -->
<toolbaritem id="button-junk">
  <toolbarbutton class="toolbarbutton-1 junk-button"
    label="&junkButton.label;"
    tooltip="&junkButton.tooltip;"
    oncommand="goDoCommand('cmd_markAsJunk')"/>
  <toolbarbutton class="toolbarbutton-1 not-junk-button"
    label="&notJunkButton.label;"
    tooltip="&notJunkButton.tooltip;"
    oncommand="goDoCommand('cmd_markAsNotJunk')"/>
</toolbaritem>
<toolbarbutton class="toolbarbutton-1" id="ButtonSpamatoStats"
  label="&stats.label;" tooltip="&stats.tooltip;"
  oncommand="spamato_showStatisticsDialog();"/>
<!-- end SPAMATO code -->
```

Damit ist die Datei “mailWindowOverlay.xul” geändert und kann wieder nach “messenger.jar/content/messenger/” zurückkopiert werden. Dazu einfach wieder die geänderte Datei per Drag&Drop ins Archiv ziehen. Zu Beachten ist dabei, dass die Originaldatei überschrieben wird. Anschliessend das Archiv wieder schliessen.

2.4.4 Änderungen in primaryToolbar.css

Die Datei “primaryToolbar.css” in “classic.jar/skin/classic/messenger/” an einen temporären Ort entpacken und zum Bearbeiten öffnen. Diese Datei ist für das Erscheinungsbild der Haupt-Toolbar von *Thunderbird* zuständig, welche unter anderem den “Junk”-Button (und künftig auch den “Statistik”-Button) enthält. In “primaryToolbar.css” werden folgende Änderungen vorgenommen:

- Zu Beginn werden die drei Definitionen des “Junk”-Button (.junk-button ...) gelöscht, da diese nachher ersetzt werden durch *Spamato*-eigene Definitionen.
- Weiter sind auch jene drei Definitionen von “.junk-button” zu löschen, welche mit dem Zusatz

```
toolbar[icons="small"]
```

beginnen. Auch sie werden nachher ersetzt.

- Nun werden die zuvor gelöschten Teile ersetzt, und zwar durch Verweise auf *Spamato*-eigene Icons, auf die nachher noch speziell eingegangen wird. Dabei wird auf spezielle Icons für “hover”- und “disabled”-Effekte verzichtet, so dass pro Button nur noch eine Definition zum Tragen kommt. Ausserdem werden Definitionen für den eigenen “Not Junk”-Button und den “Statistik”-Button hinzugefügt. Dazu wird nach diesem Kommentar:

```
/* ::::: end small primary toolbar buttons ::::: */
```

folgendes eingefügt:

```
/* ::::: SPAMATO buttons ::::: */
.junk-button {
    list-style-image: url("chrome://messenger/skin/icons/spamato_no.png");
}
.not-junk-button {
    list-style-image: url("chrome://messenger/skin/icons/spamato_ok.png");
}
toolbar[iconsize="small"] .junk-button{
    list-style-image:
        url("chrome://messenger/skin/icons/spamato_no_small.png");
}
toolbar[iconsize="small"] .not-junk-button{
    list-style-image:
        url("chrome://messenger/skin/icons/spamato_ok_small.png");
}
#ButtonSpamatoStats {
    list-style-image:
        url("chrome://messenger/skin/icons/spamato_stats.png");
}
toolbar[iconsize="small"] #ButtonSpamatoStats {
    list-style-image:
        url("chrome://messenger/skin/icons/spamato_stats_small.png");
}
/* ::::: end SPAMATO buttons ::::: */
```

Damit ist die Datei "primaryToolbar.css" geändert und kann nach "classic.jar/skin/classic/messenger/" zurückkopiert werden (alte Datei ersetzen).

2.4.5 Neue Icons in classic.jar einfügen

Da in "primaryToolbar.css" Verweise zu neue Icons verwendet wurden, müssen diese Icons nach "classic.jar/skin/classic/messenger/icons/" kopiert werden (die Dateien "spamato_no.png", "spamato_no_small.png", "spamato_ok.png", "spamato_ok_small.png", "spamato_stats.png" und "spamato_stats_small.png").

2.4.6 Änderungen in messenger.dtd

Die Datei "messenger.dtd" in "en-US.jar/locale/en-US/messenger/" an einen temporären Ort entpacken und zum Bearbeiten öffnen. Je nach installierter Sprachumgebung kann die ".jar"-Datei jedoch auch anders heissen. Grundsätzlich sollten alle Dateien geändert

werden, welche unter der oben genannten Struktur eine Datei “messenger.dtd” beinhalten.

In der Datei “messenger.dtd” sind sämtliche Tooltips und Labels der verschiedenen Objekte von *Thunderbird* abgespeichert, insbesondere der Mail-Toolbar, wo auch der “Junk”-Button definiert wurde. Da der “Statistik”-Button hinzugefügt wurde, muss dieser auch mit Label- und Tooltip-Informationen versehen werden. Folgende Änderungen sind dazu nötig:

- Unterhalb des Kommentares “Mail Toolbar” folgende Ergänzung anbringen:

```
<!ENTITY stats.label "loading..."> <!-- Statistics in SPAMATO -->
```
- Unterhalb des Kommentares “Mail Toolbar Tooltips” folgende Ergänzung anbringen:

```
<!ENTITY stats.tooltip "Statistics: #checked/#spam/#reported emails.
Press button to open config dialog.">
<!-- statistics tooltip in SPAMATO -->
```

Damit ist die Datei “messenger.dtd” geändert und kann wieder an ihren ursprünglichen Ort zurückkopiert werden (alte Datei ersetzen).

2.4.7 Änderungen in msgMail3PaneWindow.js

Die Datei “msgMail3PaneWindow.js” in “messenger.jar/content/messenger/” an einen temporären Ort entpacken und zum Bearbeiten öffnen. Darin folgende Änderungen vornehmen:

- Die Funktion “delayedOnLoadMessenger()” ist hauptsächlich für den Start von *Thunderbird* verantwortlich. Deshalb wird *Spamato* auch in dieser Funktion aufgerufen, und zwar oberhalb des Aufrufs

```
AddToSession();
```

Der Aufruf für *Spamato* lautet folgendermassen:

```
// now load Spamato
launchSpamato();
```

- Gerade das Umgekehrte geschieht in der Funktion “OnUnloadMessenger()”. Diese ist für das Beenden von *Thunderbird* wichtig, deshalb beendet folgender Aufruf auch *Spamato*, aufgerufen zu Beginn der Funktion “OnUnloadMessenger()”:

```
// shutdown SPAMATO
shutdownSpamato();
```

Auch diese Datei wird in geänderter Form an die richtige Stelle in “messenger.jar” zurückkopiert (alte Datei ersetzen).

2.4.8 Änderungen in mailWindowOverlay.js

Die Datei "mailWindowOverlay.js" in "messenger.jar/content/messenger/" an einen temporären Ort entpacken und zum Bearbeiten öffnen. In dieser Datei sind folgende Änderungen notwendig:

- Die Funktion "MsgJunkMailInfo(aCheckFirstUse)" ruft eine Informationsmeldung auf, wenn der *Thunderbird*-eigene Junkfilter zum ersten Mal aufgerufen wird. Da dieser Filter aber gar nicht benötigt wird und deshalb diese Meldung nicht erscheinen soll, muss folgendes zu Beginn der Funktion eingefügt werden:

```
pref.setBoolPref("mailnews.ui.junk.firstuse", false);  
    // This is to avoid that the infomessage  
    // for the old filter comes up.
```

Die Datei "mailWindowOverlay.js" ist somit geändert und kann wieder an ihren ursprünglichen Ort zurückkopiert werden (alte Datei ersetzen).

2.4.9 Änderungen in mailCommands.js

Die Datei "mailCommands.js" in "messenger.jar/content/messenger/" an einen temporären Ort entpacken und zum Bearbeiten öffnen. Diese Datei beinhaltet wichtige Funktionen, die auf Mails und Ordnern aufgerufen werden können. Wiederum sind einige Punkte anzupassen:

- Die Funktion "JunkSelectedMessages(setAsJunk)" ist abhängig vom Inputparameter "setAsJunk". Ist dieser *true*, so werden alle selektierten Messages als Spam angesehen und entsprechende Aktionen ausgeführt. Ist "setAsJunk" *false*, so werden die selektierten Messages als Ham angesehen. Diese Funktion entspricht genau dem Reporten/Revoken in *Spamato*. Nach

```
MarkSelectedMessagesRead(true);
```

ist folgendes aufzurufen:

```
// SPAMATO has to set the messages as Junk  
spamato_reportRevokeMessage(setAsJunk);  
  
// we can't let the thunderbird bayesian junkfilter train,  
    so we will do this  
// action in reportMessage / revokeMessage  
// gDBView.doCommand(setAsJunk ? nsMsgViewCommandType.junk  
//                               : nsMsgViewCommandType.unjunk);
```

Der letzte Befehl wird auskommentiert, da *Thunderbird* seinen eigenen Filter nicht trainieren darf. Mehr dazu später.

- Die Funktion “filterFolderForJunk()” nimmt jede Mail in einem Folder und überprüft sie auf Spam. Dabei hat der User die Möglichkeit, Mails auszulassen, deren Absenderadresse in einem “Whitelist Addressbook” stehen. Da jedoch *Spamato* über den *Ruleminator* selbst eine sehr ähnliche Funktionalität bereitstellt, wird in der *Spamato*-Implementation auf dieses Feature verzichtet. Dazu folgendes einfach auskommentieren:

```
// if enabled in the spam settings, retrieve whitelist addressbook
/* var whiteListDirectory = null;
if (spamSettings.useWhiteList && spamSettings.whiteListAbURI)
    whiteListDirectory = RDF.GetResource(spamSettings.whiteListAbURI)
        .QueryInterface(Components.interfaces.nsIAbMDBDirectory);

//Because we are using SPAMATO, this part is not interesting for us,
    so we comment it out, as well as
    we set the whiteListDirectory in 'analyzeMessage' to null */
```

- Direkt nach der obigen Änderung wird nun in einer For-Schleife jede Message aufgerufen und überprüft. Diese Überprüfung wird neu aber nicht mehr vom alten Filter, sondern von *Spamato* übernommen. Aus diesem Grund ist

```
gMessageClassifier.analyzeMessage(msgHdr, i, whiteListDirectory);
```

durch folgendes zu ersetzen:

```
checkMessage(msgHdr); // use SPAMATO to check the message!
```

Nun kann die letzte Datei “mailCommands.js” wieder an die korrekte Stelle in “messenger.jar” zurückkopiert werden (alte Datei überschreiben).

2.4.10 Fazit

Mit diesen Änderungen hat man nun ein *Thunderbird*-System, das Mails mit dem *Spamato*-Spamfilter überprüft und sie entsprechend klassifiziert. Um das Programm zu starten, genügt folgender Aufruf im *Thunderbird*-Verzeichnis:

```
./thunderbird
```

Nun sind noch drei Dinge zu tun, bevor *Spamato* seine Arbeit korrekt aufnehmen kann:

1. Es müssen alle *IMAP*-Ordner für den Offline-Gebrauch markiert und die Mails heruntergeladen werden. Dies erreicht man, indem man in *Thunderbird* einen *IMAP*-Ordner mit rechts anklickt und dann “Properties” anwählt. Dann findet sich in der Registerkarte “Offline” eine Checkbox “Select this Folder for offline use”, die man anklicken muss. Weiter muss man auch auf den Button “Download now” klicken, damit allfällige noch nicht heruntergeladene Mails ebenfalls heruntergeladen werden.

2. Man muss unter “Tools” → “Junk Mail Controls” und der Registerkarte “Adaptive Filter” erstens die Checkbox “Enable adaptive junk mail detection” anwählen und zweitens den Button “Reset Training Data” anklicken, da der Spamfilter von *Thunderbird* auf keinen Fall trainiert sein darf (mehr dazu später).
3. Man muss unter “Edit” → “Account Settings” für jeden *IMAP*-Account beim Menüpunkt “Offline & Disk Space” die Checkbox “Make the messages in my Inbox available when I am working offline” anwählen, damit Mails auf jeden Fall sofort heruntergeladen werden.

3 Aufgetauchte Probleme während der Arbeit

In diesem Kapitel will ich auf die grössten Probleme eingehen, die während dieser Arbeit entstanden sind, und angeben, ob und gegebenenfalls wie ich diese gelöst habe.

3.1 Behandlung von IMAP-Mails

IMAP-Mails werden in der Regel auf dem Mailserver behalten, und nur der Header wird heruntergeladen. Erst bei der Verwendung (z.B. Anklicken eines Mails, wodurch sein Inhalt auf einem Teil des Bildschirms angezeigt wird) wird das ganze Mail heruntergeladen. Da *Spamato* eine Pfadangabe und einen Offset für die Überprüfung eines Mails benötigt, diese Pfadangabe aber nicht die Form eines *IMAP*-Ordners haben kann (zumindest funktionierte das nicht beim Testen), kam ich zum Schluss, dass ein Mail letzten Endes immer offline gespeichert werden muss, d.h. sie muss heruntergeladen und an einem Ort auf der Festplatte gespeichert werden. Dasselbe trifft übrigens auch auf *POP3*-Mails zu: Es reicht nicht, nur den Header eines Mails herunterzuladen.

Ich unternahm mehrere Versuche, eine Mail offline zu speichern:

- Meine erste Idee war, auf einem *IMAP*-Ordner die Funktion “DownloadMessagesForOffline(messages, window)” aufzurufen. Das Array “messages” gibt dabei alle Messages an, welche heruntergeladen werden sollen. Aus irgendeinem Grund jedoch funktionierte diese Funktion nicht, auf jeden Fall wurden keine Mails heruntergeladen (“offlineSize” = 0).
- Als nächstes testete ich eine ähnlich lautende Funktion auf einem *IMAP*-Server, die aber nicht aufgerufen werden konnte, da sie scheinbar nicht für die Verwendung mit “JavaScript” freigegeben wurde.
- Dann überlegte ich mir, dass ein Mail spätestens beim Betrachten heruntergeladen werden muss, da man ja dann den Inhalt sehen kann. Deshalb versuchte ich dort einzuhaken und verfolgte den Verlauf des Programms im Code. Dieser endete jedoch irgendwann mit einem Aufruf auf der “DBView”, welche ein *XPCOM*-Objekt ist. Dies bedeutet, dass sich dahinter ein Objekt - implementiert in *C++* - befindet, in welchem die ganze Funktionalität liegt. Von hier weg konnte ich das Verhalten jedoch nicht mehr brauchbar weiterverfolgen.
- Als “endgültige” Lösung entschied ich mich daher, dass *IMAP*-Mails grundsätzlich immer heruntergeladen werden müssen (Vorgehen siehe unter “Fazit”). Mit dieser Methode funktioniert die Implementierung sehr gut.

3.2 Thunderbird-eigener Junkfilter

Ein weiteres Problem, das sich stellte, war der eingebaute Bayesian-Junk-Filter von *Thunderbird*. Dieser wird, entgegen der ersten Hoffnung, leider nicht direkt an einer Stelle aufgerufen: Für die Funktion “filterFolderForJunk”, die in “mailCommands.js” aufgerufen wird, konnte ich den Spamfilter elegant umgehen, indem ich anstatt dem Aufruf auf dem *XPCOM*-Objekt “filter-plugin;1?name=bayesianfilter” (Aufruf heisst “classifyMessage”) einfach “checkMessage” von *Spamato* verwende. Bei neu ankommenden Mails dagegen wird ein Check mit dem *Thunderbird*-Filter irgendwo direkt im *C++*-Code getätigt. Deshalb verfolge ich mit meiner Implementation folgende Strategie:

Der Filter von *Thunderbird* läuft, kurz gesagt, parallel zu *Spamato* und checkt jede eingehende Mail zuerst. Da dieser Filter ein Bayesian-Filter ist, muss er, um wirksam zu sein, zuerst trainiert werden (mittels “Junk”/“Not Junk”-Button). Da ich diese Funktionen jedoch nicht an den *Thunderbird*-Filter weiterleite, sondern nur an *Spamato*, wird der *Thunderbird*-Filter auch nicht trainiert. Deshalb wird der Filter alle eingehenden Mails entweder als “Ham” oder als “Unknown” interpretieren. Als nächstes kommt dann *Spamato* an die Reihe, das die Mail wirklich klassifiziert.

Optimal wäre an dieser Stelle natürlich, den Spamfilter von *Thunderbird* ganz aus dem Code zu nehmen. Da ist jedoch, wie oben erwähnt, das Problem, dass dieser an mehreren Stellen verwendet wird, und man nicht sicher sein kann, alle diese Stellen aufgespürt zu haben. Eine zweite, sichere Variante würde darin bestehen, dem *Thunderbird*-Spamfilter irgendwo angeben zu können, dass jede Mail als “Ham” betrachtet wird.

Meine Strategie funktioniert soweit gut, sie basiert jedoch auf der Annahme, dass der *Thunderbird*-Spamfilter keine “Spam”-Klassifikation vorwegnimmt.

3.3 Dokumentation von Thunderbird

Das letzte grössere Problem war, dass der Code von *Thunderbird* extrem schlecht dokumentiert ist, sowohl im Code selbst als z.B. auch die Online-API-Dokumentation auf <http://www.xulplanet.com>. Oft ist gar nicht ersichtlich, was z.B. eine Funktion für Parametertypen erwartet oder was eine Funktion genau tut. Dies machte auch viele Umwege nötig, die möglicherweise ganz elegant mit *Thunderbird*-eigenen Mitteln zu realisieren gewesen wären.