

A Google Sidebar Panel for Spamoto



Roman Fuchs

Semesterarbeit

15. Oktober 2006

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

Inhaltsverzeichnis

1	Einleitung	2
1.1	Ziel der Arbeit	2
1.2	Übersicht	2
2	Google Desktop	3
2.1	Development Model 4.0 für skriptbasierte Gadgets	3
2.2	Kommunikation	4
3	Implementation	5
3.1	Architektur	5
3.2	Spamato-Plugin	5
3.3	Kommunikation	6
3.3.1	XML-Antwort	6
3.4	Design des Gadgets	7
4	Fazit	9
4.1	Erfahrungen	9

1 Einleitung

Spamato ist ein erweiterbares, kollaboratives Spam-Filter-System der Distributed Computing Group der ETH Zürich [1]. Es kombiniert verschiedene Spam-Filter-Techniken, um Spam wirksam zu filtern. Es beinhaltet auch weitere Komponenten, die dem Benutzer das Filtern erleichtern. Eine dieser Komponenten ist die Filter-History, welche die detaillierten Filterergebnisse der Nachrichten anzeigt, die Spamato überprüft hat.

Auch der Google Desktop erfreut sich in letzter Zeit wachsender Beliebtheit. Zentrale Elemente davon sind die Desktop-Suche und die Google Sidebar. Erweitern kann man den Google Desktop einerseits durch die Entwicklung von Indexing-Plugins, die der Desktop-Suche ermöglichen, Dateien in neuen Formaten zu durchsuchen. Die Google Sidebar andererseits lässt sich mit Panels (neuerdings „Gadgets“ genannt) erweitern, die personalisierte Informationen aus dem Internet abrufen, den Benutzer über wichtige Ereignisse informieren und ihm ermöglichen, in der Sidebar spezifische Benutzeraktionen vorzunehmen. Typische Beispiele sind personalisierte Nachrichten, RSS-Feeds, Wetterinformationen, Börsenkurse oder ein System-Monitor.

1.1 Ziel der Arbeit

Ziel meiner Arbeit war es, eine Brücke zwischen Spamato und der Google Sidebar zu erstellen, um sich die Ergebnisse der Spamfilter anzeigen zu lassen und einen direkten Zugriff auf die Filter-History zu erhalten. Dazu musste einerseits ein Spamato-Plugin entwickelt werden, das die gewünschten Ergebnisse von Spamato sammelt und an die Google-Desktop-Komponente weitergibt. Andererseits musste ein Google-Gadget entwickelt werden, das die Ergebnisse in der Sidebar präsentiert und dem Benutzer ermöglicht, sich die Filter-History direkt anzeigen zu lassen. Dieses sollte auch das bereits existierende E-Mail-Gadget der Sidebar ersetzen. Eine zentrale Bedeutung kam der Kommunikation zwischen den beiden Plugins zuteil.

1.2 Übersicht

Nach dieser Einleitung wird in Kapitel 2 der Google Desktop eingeführt. Ein Schwerpunkt wird dabei auf die Entwicklung von skriptbasierten Gadgets gelegt. Kapitel 3 erläutert die Implementation der beiden Plugins. Insbesondere wird das gewählte Kommunikationsmodell erläutert. Abschliessend wird in Kapitel 4 noch ein Fazit gezogen und Verbesserungsmöglichkeiten aufgezeigt.

2 Google Desktop

Der Google Desktop ist noch in der Entwicklung und hat sich in der Zeit dieser Semesterarbeit auch weiterentwickelt von Version 3 auf Version 4. Desktop Gadgets können prinzipiell in jeder Programmiersprache geschrieben werden, die COM und XML unterstützt. Praktisch gesehen wurden bisher aber nur Gadgets in C++, C#, JavaScript und VBScript geschrieben. Weil Gadgets in C# das .NET-Framework laden müssen und somit zusätzlichen Speicher benötigen, hat Google jedoch bis heute noch kein Gadget in C# akzeptiert und auf ihrer Webseite zum Download veröffentlicht. Im neusten Developer Guide wurden die Beispiele in C# daher auch nicht mehr aufgeführt. Die Entwicklung in Skriptsprachen wurde mit dem Development Model 4.0 deutlich vereinfacht und soll im folgenden kurz vorgestellt werden.

2.1 Development Model 4.0 für skriptbasierte Gadgets

Nachdem vor dem Development Model 4.0 [3] noch der Microsoft Windows Installer (MSI) verwendet wurde, bestehen die neuen Google-Gadgets nur noch aus einem ZIP-Archiv mit dem Suffix .gg. Ein skriptbasiertes Gadget besteht aus folgenden Komponenten:

- **gadget.gmanifest** Enthält Metadaten, die das Gadget im XML-Format beschreiben.
- **main.xml** Definition der Hauptansicht des Gadgets. Ein *view*-Objekt besteht aus dem UI und den Objekten, die zusammen ein sichtbares Fenster des Gadgets ergeben.
- **options.xml** Definition der Optionenansicht des Gadgets (sofern erforderlich).
- Sämtliche Skriptdateien mit dem Code, der zum Implementieren der Funktionalität des Gadgets erforderlich ist.
- Alle benötigten Bilddateien.

Um mehrere Sprachen zu unterstützen, besteht ausserdem die Möglichkeit, sprachspezifische Strings in einer Datei strings.xml in Unterordnern abzulegen, die nach der jeweiligen Sprach-ID benannt sind.

Das Erscheinungsbild des Gadgets wird mit einem *view*-Objekt definiert. Dieses kann aus verschiedenen UI-Elementen bestehen, die im folgenden kurz aufgelistet werden:

- **button** Eine Schaltfläche.
- **checkbox** Ein Kontrollkästchen.
- **object** Ein eingebettetes ActiveX-Steuerelement ohne Fenster.
- **contentArea** Enthält eine Sammlung von *ContentItem*-Objekten.

- **div** Hiermit werden die Farben oder Bilder für den Elementhintergrund festgelegt.
- **edit** Ein Bereich, in dem der Benutzer des Gadgets den Textinhalt bearbeiten kann.
- **img** Ein Bild.
- **label** Ein Textlabel.
- **a** Ein HTML-Link (Es können jedoch nur URLs vom Typ `http:`, `https:` und `ftp:` geöffnet werden.)
- **progressBar** Eine Statusanzeige.

Ein vielverwendetes UI-Element ist die *contentArea*. Es ist im Wesentlichen eine Datenstruktur für *ContentItems* und bietet ausschliesslich Methoden zum Hinzufügen und Entfernen solcher Elemente an.

Das *ContentItem*-Objekt besteht aus fixen Attributen, deren wichtigste hier aufgelistet sind:

- **time_created** Erstellungszeit des Elements.
- **heading** Der angezeigte Titel des Elements.
- **source** Die angezeigte Quelle des Elements.
- **snippet** Der in der Detailansicht angezeigte Ausschnitt des Elements.
- **open_command** URL/Dateipfad, der angezeigt wird, wenn der Nutzer das Element öffnet/auf das Element doppelklickt.
- **layout** Layout des Elements (3 fixe Formate sind vorgegeben).
- **tooltip** Tooltipp-Text.

Daneben besitzt das *ContentItem*-Objekt einige Ereignishandler. Der wichtigste davon ist *onDetailsView*, der aufgerufen wird, bevor die Detailansicht für das entsprechende Element angezeigt wird.

2.2 Kommunikation

Die Kommunikation von Gadgets nach aussen ist prinzipiell nur über *XMLHttpRequest*-Objekte möglich. Die meisten Gadgets verwenden diese Möglichkeit, um einen GET-Request an einen Webserver zu richten und die erhaltene Information zu verwerten.

Identische Gadgets können ausserdem untereinander kommunizieren, sofern alle Nutzer der kommunizierenden Gadgets bei Google Talk angemeldet sind. Dies wird in erster Linie für Spiele wie Tic Tac Toe verwendet.

3 Implementation

3.1 Architektur

Um die gewünschten Funktionalitäten zu implementieren, wurden sowohl Spamoto wie auch die Google Sidebar mit geeigneten Plugins erweitert. Das Spamoto-System wurde über die Plugin-Architektur [2] mit einem *GoogleSidebarPlugin* erweitert. Nach abgeschlossener Spam-Filterung übergibt Spamoto dem Plugin ein Objekt *FilterProcessResult*, das die Inhalte des Mails sowie die Filterergebnisse beinhaltet. Das Plugin bereitet die Ergebnisse auf und überträgt sie anschließend zur Google Sidebar. Für die Google Sidebar wurde ein Spamoto-Gadget entwickelt, das die Resultate in XML-Form entgegennimmt und sie anschließend in der Sidebar anzeigt. Die Übertragung der Mails läuft über einen lokalen Socket.

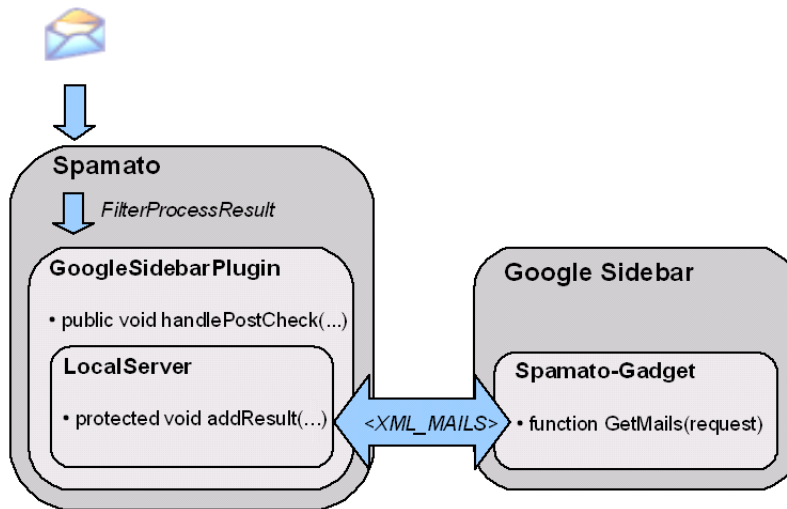


Abbildung 1: Architektur der beiden Plugins

3.2 Spamoto-Plugin

Für die Implementation des Spamoto-Plugins wurde Java 1.5.0 verwendet [4]. Das Plugin erbt vom Interface *PostChecker* und übergibt dem Spamoto-LocalServer die *FilterProcessResult*-Objekte. Die wichtigsten Methoden und Attribute des Spamoto-LocalServers sind folgende:

```
public class LocalServer implements Runnable, Disposable {
    private Vector<FilterProcessResult> newMails = new Vector<FilterProcessResult>();
    private Vector<FilterProcessResult> initialMails = new Vector<FilterProcessResult>();

    protected void addResult(FilterProcessResult result) {...}
    private String getXMLMail(FilterProcessResult filterResult) {...}
    private String getNewMails() {...}
    private String getInitialMails() {...}
}
```

In der Methode *addResult* werden die *FilterProcessResult*-Objekte in den beiden *Vector*-Attributen gespeichert. Die Methode *newMails* enthält jeweils die Mails,

die bisher noch nicht an die Sidebar übertragen wurden, während *initialMails* jeweils die 10 letzten erhaltenen Mails speichert.

Der LocalServer des Spamato-Plugins verarbeitet zwei Requests: „initial“ und „newmails“. Der erste wird vom Sidebar-Plugin verwendet, wenn er neu initialisiert wird, um die letzten erhaltenen Mails anzufordern. Der zweite Request hingegen fragt nur die neuen Mails ab. Die Methode *getXMLMail* liest die gewünschten Informationen aus den Mails aus und bringt sie in die XML-Struktur, die für die Übertragung zum Google-Gadget verwendet wird. Auf die Details der Kommunikation geht der nächste Abschnitt ein.

3.3 Kommunikation

Die Kommunikation zwischen dem Spamato-Plugin und dem Google-Gadget läuft über *XMLHttpRequests*. Es ist die vorgesehene Kommunikationmöglichkeit von Google-Gadgets, die häufig Informationen von Webservern in XML-Form abrufen. Dazu beinhaltet das Spamato-Plugin einen *LocalServer*, der Anfragen entgegennimmt und eine XML-Antwort zurückschickt. Es wird das Pull-Kommunikationsmodell verwendet, und das Google-Gadget ruft die Resultate periodisch ab.

```
function GetMails(request) {
    // send request to spamato for emails
    var http = new XMLHttpRequest();
    http.onreadystatechange = OnData;
    var url = "http://localhost:7733/" + request;
    http.open("GET", url, true);
    http.setRequestHeader("If-Modified-Since", "Sat, 1 Jan 2000 00:00:00 GMT");
    http.send();

    function OnData() {
        if (http.readyState == 4) {
            var doc = new DOMDocument();
            doc.loadXML(http.responseText);
            // parse the xml data received
            ...
        }
    }
}
```

3.3.1 XML-Antwort

Die XML-Antwort kann mehrere Mails beinhalten und ist im folgenden dargestellt:

```
<?xml version="1.0" encoding="UTF-8"?>
<spamato>
  <mail>
    <id>MAIL_ID</id>
    <port>CONFIG_PORT</port>
    <subject>SUBJECT</subject>
    <sender>SENDER</sender>
    <result>FILTERRESULT</result>
    <ishtml>IS_HTML</ishtml>
    <date>DATE</date>
    <text>MESSAGE</text>
  </mail>
  <mail>
    ..
  </mail>
  ..
</spamato>
```

Die XML-Antwort beinhaltet folgende Daten:

- MAIL_ID Die Mail-ID, die benutzt wird um die Filter-History zu öffnen.
- CONFIG_PORT Der aktuelle Config-Port von Spamato.
- SUBJECT Der Betreff des Mails.
- SENDER Der Absender des Mails.
- FILTERRESULT Die Resultate der Spamfilter (z.B. "2/6").
- IS_HTML Ein Flag, das anzeigt, ob das Mail in HTML geschrieben ist.
- DATE Das Sendedatum des Mails (Format: "TT.MM.YY hh:mm:ss").
- MESSAGE Die Nachricht des Mails.

Um die Verarbeitung der XML-Daten nicht zu behindern, werden die Zeichen '<', '>', sowie '&' als '<', '>' und '&' kodiert. Und damit beispielsweise auch Umlaute und das europäische Währungszeichen unterstützt werden, wird die Antwort in UTF-8 kodiert.

3.4 Design des Gadgets

Das Spamato-Gadget besteht aus einer Liste von erhaltenen E-Mails. In der Hauptansicht werden Filterresultat, Betreff, Absender und Zeitpunkt des Versandes angezeigt. Über der Liste gibt es noch einen Direktlink, der die Filter-History öffnet und einem die Möglichkeit eines Report oder Revoke bietet.

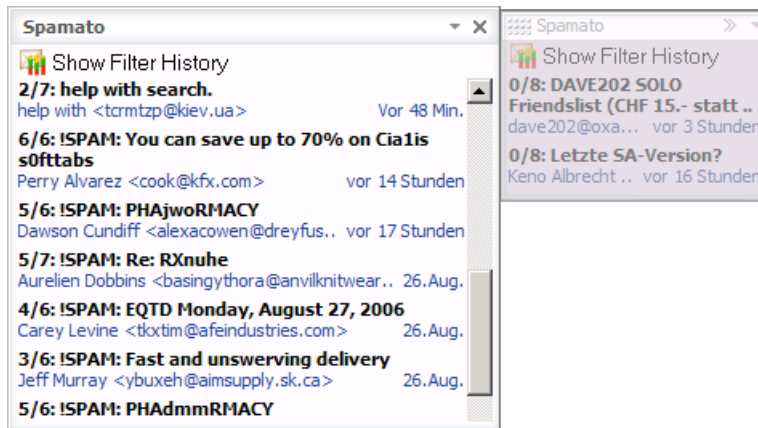


Abbildung 2: Spamato-Gadget mit erweiterter Ansicht.

Ein einfacher Klick auf ein *ContentItem* öffnet die *DetailsView*. Dort wird zusätzlich die Nachricht des Mails entsprechend dem Flag *IS_HTML* in HTML oder reiner Textform angezeigt.

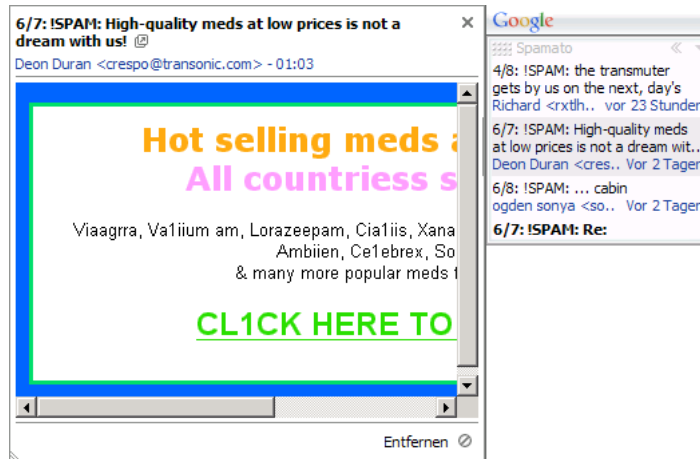


Abbildung 3: Spamato-Gadget mit Detailansicht einer HTML-Nachricht.

Ein Doppelklick auf ein *ContentItem* öffnet die Filter-History des entsprechenden Mails, in dem die genaueren Ergebnisse der Spamfilter dargestellt werden.

4 Fazit

Meine Arbeit hat gezeigt, dass der Informationsaustausch zwischen Spamato und der Google-Sidebar mit Hilfe von XMLHttpRequests einfach und effizient möglich ist. Das Kommunikationsmodell mit XML-Struktur ist sehr offen.

Schwierigkeiten ergaben sich jedoch bei der Gestaltung des Google-Gadgets. Die Komponente *contentArea* ist momentan noch kaum erweiterbar. So wäre es wünschenswert, dass man in der Detailansicht direkt die Möglichkeit hätte, eine E-Mail mit einem *Report* als Spam zu melden oder eine Spam-Erkennung durch ein *Revoke* zu widerrufen. Die momentane Lösung führt über einen Link zur Filter-History, wo man diese Möglichkeiten erhält.

Zudem sind die Einträge der *contentArea* leider nicht persistent. Da die Sidebar beim Minimieren alle Gadgets entlädt und beim Wiederherstellen wieder neu lädt, gehen die Einträge verloren. Die letzten zehn erhaltenen Mails werden daher beim Spamato-Plugin gespeichert und beim Laden des Google-Gadgets erneut übertragen. Dies funktioniert natürlich nur, wenn Spamato währenddessen nicht beendet wurde.

Ein weiteres Problem war, dass das Google Desktop API [3] nicht vollständig dokumentiert war. Ähnliche Probleme hatten auch andere Entwickler von Google Gadgets, und so war das Google Developer Forum [5] eine Anlaufstelle für entsprechende Fragen.

4.1 Erfahrungen

Die Einarbeitung ins Spamato-System verlief auch dank der Unterstützung des Betreuers ziemlich reibungslos. Zu Beginn hatte ich eher etwas Mühe, mich bei der Pluginentwicklung für die Google Sidebar zurechtzufinden. Die Kommunikation zwischen den Plugins hat mich ziemlich lange beschäftigt. Sie war letztlich jedoch einfacher als erwartet. Oftmals waren es aber sehr kleine Probleme, die Zeit und Geduld erforderten.

Zur Arbeitshaltung habe ich gelernt, wie wichtig es ist, eine klare Zeitplanung zu erstellen und sich daran zu halten. Nicht zuletzt durch zwei projektorientierte Vorlesungen, die mich während des Semesters ziemlich beanspruchten, geriet ich ein wenig in Verzug und benötigte für die Arbeit mehr Zeit als geplant.

Literatur

- [1] Keno Albrecht, Nicolas Burri and Roger Wattenhofer. Spamato An Extendable Spam Filter System.
2nd Conference on Email and Anti-Spam (CEAS), Stanford University, Palo Alto, California, USA, Juli 2005.
- [2] Remo Meier. Spamato Plug-in Architecture.
Semesterarbeit, ETH Zürich, 2005.
- [3] Google Desktop API
<http://desktop.google.com/script.html>.
- [4] Sun Microsystems. Java 2 Platform, Standard Edition, Version 5.0.
<http://java.sun.com/j2se/1.5.0>.
- [5] Google Desktop Developer Forum
<http://groups.google.com/group/Google-Desktop-Developer>.