JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Design und Implementierung eines Frameworks zur Integration von sicherheitsrelevanten Erweiterungen in Mailclients

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Mag. rer. soc. oec

im Diplomstudium

WIRTSCHAFTSINFORMATIK

Eingereicht von:
Christian Haider

Angefertigt am:
Institut für Informationsverarbeitung und Microprozessortechnik

Betreuung:
o. Univ. Prof. Dr. Jörg R. Mühlbacher

Mitbetreuung:
Dr. Andreas Putzinger

Linz, Juni 2008

Danksagung

An dieser Stelle möchte ich einigen Personen speziell danken, die mir einerseits bei meinem Studium und insbesondere bei der Anfertigung dieser Magisterarbeit eine große Hilfe waren.

Allen voran bedanke ich mich bei meinen Eltern, meinem Bruder Emanuel und seiner Frau Renate und meinen zukünftigen Schwiegereltern, ohne deren Hilfe und Unterstützung dieses Studium nicht möglich gewesen wäre. Außerordentlicher Dank gebührt meiner Verlobten Susanne für ihre Geduld und aufmunternden Worte.

Weiters danke ich allen Institutsmitarbeitern, welche mich hierbei unterstützt haben. In erster Linie gebührt dies natürlich meinen Betreuern, die durch ihre fachliche Expertise eine große Unterstützung für mich waren. Ein besonderer Dank gilt Dr. Andreas Putzinger, der stets ein offenes Ohr für meine Anliegen hatte und mir zahlreiche hilfreiche Anregungen gab.

Kurzfassung

Beim Entwurf von vom Mozilla Projekt abstammenden Mailclients wurde eine Designschwäche übersehen, welche die Einbindung von sicherheitsrelevanten Erweiterungen erschwert. Die Implementierung eines Frameworks bietet eine Möglichkeit zur Behebung dieser Schwäche und erlaubt zusätzlich, die Entwicklung von Erweiterungen zu beschleunigen.

Durch das Anbieten dieses Frameworks soll erreicht werden, dass Entwicklern, die gute Ideen für nützliche Erweiterungen haben, diese aber aufgrund fehlenden Wissens nicht effizient umsetzen können, eine Möglichkeit gegeben wird, diese in kürzerer Zeit zu realisieren. Es kann davon ausgegangen werden, dass durch dieses Vorgehen die Anzahl an verfügbaren sicherheitsrelevanten Erweiterungen entscheidend gesteigert wird

Abstract

There are some shortcomings in the design of Mozilla mail clients that hamper the integration of security-relevant add-ons. The implementation of a framework shall help to overcome these shortcomings and in addition allow the acceleration of the development process of add-ons.

The Framework shall enable developers with good ideas but missing consolidated knowledge to safe time and effort implementing their concepts of useful add-ons. It can be assumed that the facilitation of the development process will lead to an increase in the number of available security-relevant add-ons.

INHALTSVERZEICHNIS

1 EINLEITUNG 6
1.1 PROBLEMBESCHREIBUNG 6
1.2 ZIEL DER DIPLOMARBEIT 7
1.3 STRUKTUR DER DIPLOMARBEIT 7
2 AUSGANGSSITUATION 8
2.1 MOZILLA PROJEKT 8
2.2 THUNDERBIRD 9
2.3 BESTEHENDE LÖSUNGEN 12
2.4 GRENZEN DER BESTEHENDEN LÖSUNGEN 16
2.5 LÖSUNGSANSATZ 18
3 ERWEITERUNGEN VON THUNDERBIRD 21
3.1 EINFÜHRUNG MOZILLA PROGRAMMIERUNG 22
3.1.1 XUL (XML User Interface Language) 24
3.1.2 CHROME 26
3.1.3 Overlays 27
3.1.4 Skins, Styles and Themes 30
3.1.5 Localization 30
3.1.6 Preferences 31
3.1.7 Install.rdf und chrome.manifest 33
3.1.8 Packaging 37
3.1.9 XPCOM 38
3.1.9.1 Interface 38
3.1.9.2 Components 39
3.1.9.3 XPIDL 41
3.1.9.4 Factory 42
3.1.9.5 Module 43
3.2 ENVIRONMENT FÜR EINFACHE PROJEKTE 44
3.3 ENVIRONMENT FÜR KOMPLEXE PROJEKTE 45
3.3.1 Installationen 46
3.3.2 CVS 46
3.3.3 Erstes Build 48
3.3.4 Build-Prozess mit Erweiterung 50
4 DAS JACKF PROJEKT 54
4.1 ÜBERBLICK 54
4.2 HERAUSFORDERUNG 56
4.2.1 Einfachheit 56
4.2.1.1 Konfiguration an einem zentralen Punkt 56
4.2.1.2 Übersichtliche Darstellung 57
4.2.1.3 Automatisches Update 58
4.2.1.4 Einfach zu verwendende Schnittstellen 59
4.2.2 Performance 61
4.2.2.1 Threading 62
4.2.2.2 Terminieren von Threads 63
4.2.2.3 Implementieren der Kernfunktionen in C++ 64
4.2.2.4 Gemeinsame Aufgaben kombinieren 65
4.2.3 Plattformunabhängig 65
4.2.4 Zwischenspeichern von Analyseergebnissen 66
4.3 BENUTZERHANDBUCH 68
4.3.1 Installation 68
4.3.2 Konfiguration 69
4.4 ARCHITEKTUR 71
4.4.1 Display Klassen 73
4.4.2 Datenbank Klassen 77

4.4.3 *Konfigurationsklassen* ... 79
4.4.4 *JackF Kernstück* ... 81
4.4.5 *Worker Klassen* ... 84
4.4.6 *Hilfsklassen* ... 86
4.4.7 *Zusätzliche Diagramme* ... 87
4.4.7.1 JackFExtensionRegistration ... 87
4.4.7.2 Analyseabfrage auf höchstem Detaillierungsgrad ... 88
4.5 DESIGNENTSCHEIDUNGEN UND AUSGEWÄHLTE CODEFRAGMENTE ... 89
4.5.1 *Analyseergebnisse in einer Spalte in der Nachrichtenübersicht* ... 89
4.5.2 *Workerthread Wiederverwendung* ... 92
4.5.3 *Zwischenspeicher zum Reduzieren von RDF Manipulationen in nsIJackFDisplayManager* ... 93
4.5.4 *nsIJackFScoreDB* ... 95
4.6 SCHREIBEN EINES PLUGGIES – ENTWICKLERHANDBUCH ... 96
4.7 PROBLEME ... 99
4.7.1 *Bugs im Mozilla Projekt* ... 99
4.7.1.1 Bug 406414 ... 100
4.7.1.2 Bug 400627 ... 100
4.7.1.3 Bug 419192 ... 101
4.7.2 *Releasewechsel von Thunderbird* ... 101
4.7.2.1 Version 1.5 → 2.0 ... 101
4.7.2.2 Version 2 → 3 ... 102

5 ZUSAMMENFASSUNG ... 104

6 ABBILDUNGS- UND TABELLENVERZEICHNIS ... 106

7 LITERATURVERZEICHNIS ... 108

8 APPENDIX ... 110

8.1 QUELLCODE EXAMPLE ... 110
8.1.1 *contents.rdf (Seite 28)* ... 110
8.1.2 *example.xul (Seite 31 und 33)* ... 110
8.1.3 *JS Beispiel Component* ... 111
8.1.4 *C++ Beispiel Component* ... 112
8.1.4.1 nsExampe.h ... 112
8.1.4.2 nsExampe.cpp ... 112
8.1.4.3 Examplemodule.cpp ... 113
8.1.5 *jackftemplate.js (Seite 98)* ... 113
8.2 BATCH DATEI FÜR .XPI PAKET ... 116

9 EIDESSTATTLICHE ERKLÄRUNG ... 117

10 CURRICULUM VITAE ... 118

1 Einleitung

1.1 Problembeschreibung

Computer sind täglich einer Vielzahl an Gefahren ausgesetzt. Ein Großteil dieser Gefahren wird erst durch die Vernetzung von Computern und die Verwendung des Internets akut. Als Mailclient stellt Thunderbird eine Schnittstelle zwischen Anwender und potentiell gefährlichen, aus dem Internet stammenden, Inhalten dar. Nachrichten können mit Viren oder Spyware verseucht sein, mittels Phishing versuchen, einem Anwender vertrauliche Informationen zu entlocken, oder in die Kategorie von unerwünschtem Spam fallen. Da viele Anwender Nachrichten ohne vorherige Prüfung auf Schadsoftware abrufen und ungeachtet des möglichen Inhaltes öffnen, wäre es wünschenswert, dass Mailclients eine erste Barriere für Schadsoftware bereitstellen. [Sun05] vertritt die Meinung, dass mit zunehmender Verbreitung von Firefox auch Spyware und Adware speziell für diesen entwickelt werden wird. Kann diese Aussage auf Thunderbird übertragen werden, bedeutet dies, dass in absehbarer Zeit nicht mehr nur maßgeschneiderte Angriffe auf Microsoft Produkte, sondern auch auf Thunderbird auftreten werden.

Thunderbird bietet die Möglichkeit, seine Grundfunktionalität durch Installieren von Erweiterungen zu ergänzen. Die derzeit erhältlichen Erweiterungen sind in ihrer Anzahl gering und decken nicht die wünschenswerte Breite ab. Der Grund dafür könnte sein, dass das Entwickeln von Erweiterungen für einen Anwender trotz ausreichender Programmierkenntnisse einen hohen Einarbeitungsaufwand darstellt. Durch eine eingebaute Strukturschwäche in Thunderbird wird das Entwickeln von Erweiterungen, welche eine ressourcenintensive Analyse einer Nachricht durchführen, zusätzlich erschwert. Um die Anzahl der zur Verfügung stehenden Erweiterungen zu erhöhen, ist es notwendig, dass Probleme und Schwächen von Erweiterungen auf einer abstrakten Ebene gelöst werden. Damit können alle Nebenaufgaben einer Erweiterung, darunter fallen Aufgaben wie das Zwischenspeichern von Analyseergebnissen und das Darstellen von Ausgaben, durch ein Framework realisiert werden. Der Entwickler einer Erweiterung kann sich dadurch seiner eigentlichen Aufgabe bzw. Idee widmen und die Kernfunktion der Erweiterung implementieren.

1.2 Ziel der Diplomarbeit

Das Ziel dieser Diplomarbeit ist es, das Framework namens JackF, welches im Rahmen dieser Diplomarbeit entwickelt wurde, vorzustellen. Das Framework dient zur Integration von sicherheitsrelevanten Erweiterungen in Mailclients. Zusätzlich wird dem Leser das notwendige Wissen vermittelt, welches zum Implementieren von Erweiterungen für von Mozilla abstammende Anwendungen benötigt wird. Basierend auf diesem Grundgerüst werden die Einschränkungen von „normalen" Erweiterungen aufgezeigt und dem Leser erläutert, warum die Verwendung des JackF Frameworks für ihn entscheidende Vorteile bietet.

1.3 Struktur der Diplomarbeit

Die Diplomarbeit ist in 3 Teilbereiche gegliedert. Im ersten Hauptkapitel wird einleitend das Mozilla Projekt sowie das Konzept, welches sich hinter dem Mailclient mit dem Namen Thunderbird verbirgt, vorgestellt. Im Anschluss wird ein Überblick über bestehende Erweiterungen für Thunderbird mit sicherheitsrelevantem Aspekt gegeben. Basierend auf diesen Erweiterungen werden die Schwächen im Design von Thunderbird und die Grenzen für „normale" Erweiterungen aufgezeigt. Abschließend wird in diesem ersten Hauptkapitel noch grob der Lösungsansatz, der mit JackF verfolgt wird, beschrieben.

Um die Strukturschwäche von Thunderbird genauer beleuchten zu können, beginnt das zweite Hauptkapitel mit einem Überblick über die grundlegende Technik, die zum Entwickeln von Erweiterungen für Mozilla angewandt wird. Zusätzlich zu dieser theoretischen Einführung wird das Einrichten zweier Entwicklungsumgebungen erläutert. Erstere bietet die Möglichkeit Erweiterungen, die mit JS implementiert werden, zu erstellen. Für anspruchsvollere Erweiterungen, die in C++ programmiert werden oder zum Kompilieren von Thunderbird selbst, ist die zweite Entwicklungsumgebung gedacht.

Im abschließenden dritten Hauptkapitel, welches den Großteil der Diplomarbeit ausmacht, wird das JackF Framework und dessen Design vorgestellt. Neben den Herausforderungen, welche an das Framework gestellt werden und dessen Design, wird in einem Benutzerhandbuch dessen Handhabung erklärt. Entwicklern, die das Framework einsetzen möchten, wird in einem Entwicklerhandbuch die Verwendung erläutert. Abschließend werden einige Probleme angeführt, die während der Implementierung bewältigt werden mussten.

2 Ausgangssituation

2.1 Mozilla Projekt

Der Grundstein für das heutige Mozilla-Projekt wurde am 22. Jänner 1998 gelegt, als Netscape Corporation den Quellcode seines hauseigenen Webbrowsers Netscape 5.0 unter der NPL (Netscape Public License) Lizenz veröffentlichte. Damit garantierte das stark geschwächte, mit Microsoft in Konkurrenz stehende Unternehmen das Weiterbestehen eines freien Webbrowsers. Die nun seit mehr als 10 Jahren bestehende, sehr aktive Entwicklergemeinde konzentrierte sich zu Beginn auf die Weiterentwicklung der Mozilla Application Suite. Mit der Kombination eines Webbrowsers, Mailclients, IRC-Clients und HTML-Editors in einem Produkt wurde dasselbe Ziel wie von Netscape Communications verfolgt.

Erst 2002 entschieden sich die Entwickler, die gemeinsame Codebasis in die Module

- Webbrowser – Phoenix, heute als Firefox bekannt
- Emailclient – Thunderbird
- Kalender – Sunbird
- HTML-Editor – Nvu

aufzuteilen, um diese flexibler und besser entwickeln zu können. Durch die Modularisierung konnte auch erreicht werden, dass die einzelnen Anwendungen weniger Ressourcen als das gesamte Paket benötigen.

Obwohl der Quellcode der einzelnen Mozilla-Projekte öffentlich zugänglich ist, sind mit dessen Verbreitung und Weiterentwicklung gewisse Pflichten verbunden. Beispielsweise sind die Logos und Namen urheberrechtlich geschützt und dürfen nicht ohne Zustimmung in veränderten Versionen genutzt werden. Dies wird damit begründet, dass Mozilla für höchste Qualität steht, und die Anwender durch diese Einschränkung auf das Symbol vertrauen können.

Mozilla unterscheidet 3 Varianten von Quellcodeänderungen:

- *Official Localized Releases:* Diese Releases dürfen sich als offizielle Versionen - von zum Beispiel Thunderbird - bezeichnen und dessen Logo und Namen verwenden. Sie

müssen allerdings von Mozilla anerkannt werden und sich an genauen Richtlinien orientieren.

- *Community Editions:* Die Richtlinien, durch welche diese Releases geregelt sind, sind nicht so strikt, weshalb sie keinen offiziellen Status erhalten, und die von Mozilla registrierten Logos und Namen nicht dafür verwendet werden dürfen. Weiters muss das Produkt als Community Edition gekennzeichnet werden.
- *Serious Modifications:* Sollte der Quellcode eines bestehenden, anerkannten Programms signifikant geändert worden sein, darf dieses nicht unter einem von Mozilla registrierten Namen veröffentlicht werden. Überdies sollte das Produkt keine Bezeichnungen wie zum Beispiel „basierend auf Mozilla Firefox" enthalten. Um diesem Problem aus dem Weg zu gehen, wird geraten, Bezeichnungen wie „basierend auf Mozilla Technologie" heranzuziehen.

Aufbauend auf dem Quellcode von Mozilla ist es weiters möglich, eigene Anwendungen zu entwickeln. Eine gute Basis dafür bietet der XUL-Runner, welcher die wichtigsten Teile einer webbasierten Anwendung bereitstellt, und dessen Quellcode für die Betriebssysteme Windows, Linux und MAC OSX verwendbar ist [Ste07].

2.2 Thunderbird

Der immer größere Verbreitung findende Mailclient wurde das erste Mal am 28. Juli 2003 in der Version 0.1 veröffentlicht. Auch wenn es sich dabei um ein lauffähiges Programm handelte, war dieses nicht zur Benutzung durch Endanwender gedacht, sondern sollte eine Basis für Entwickler schaffen.

[LWW05] nennt als einen wesentlichen Erfolgsfaktor von Software, ob diese als Open Source verfügbar ist. Thunderbird teilt diesen Vorteil mit anderen dem Mozilla Projekt abstammenden Produkten. Mit über 62 Millionen heruntergeladen Kopien seit dem Erscheinen der Version 1.0 im Dezember 2004 stellt dieser eine Konkurrenz für Microsofts Mailclients dar.

Die Benutzeroberfläche ist den Produkten Outlook und Outlook Express von Microsoft dem Aussehen nach sehr ähnlich. Anwendern, die bereits einen dieser Mailclients verwendet haben, wird der Umstieg auf Thunderbird keine großen Schwierigkeiten bereiten. Sowohl Nachrichten als auch das Adressbuch und Mailkontoeinstellungen können durch Thunderbird importiert und weiterbenutzt werden.

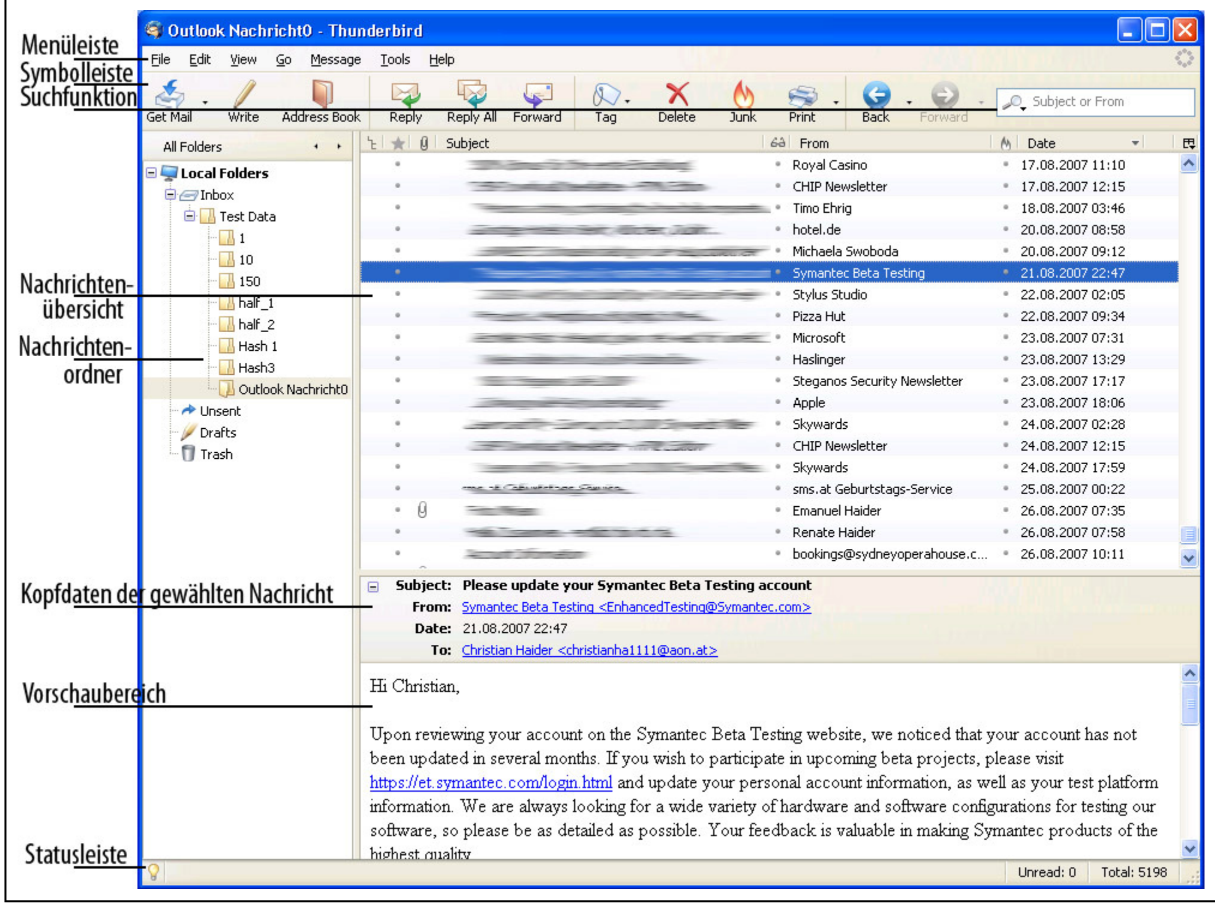

Abbildung 2.1: Thunderbirds Benutzeroberfläche

Das primäre Ziel von Thunderbird ist es, einen einfach zu benutzenden, sicher und stabil arbeitenden Mailclient zur Verfügung zu stellen. Viele Programme werden heute mit dem Gedanken entwickelt, ein Grundgerüst bereitzustellen, das durch Erweiterungen, den sogenannten PlugIns, in seiner Funktionalität ergänzt werden kann. Auch bei der Entwicklung von Thunderbird wurde dieses Konzept aufgegriffen. So ist etwa der in Thunderbird integrierbare Terminkalender als Erweiterung realisiert. Diese Erweiterungen können als `.xpi` Dateien im Internet zum Download angeboten werden und beinhalten alle für die Installation erforderlichen Daten. Der Aufbau einer solchen `.xpi` Datei wird in Kapitel *3.1.8 Packaging* näher beschrieben. Um den Anwendern die Installation der für sie brauchbaren Erweiterungen zu erleichtern, bietet Thunderbird mehrere Möglichkeiten. Am einfachsten ist es, in der Menüleiste `Tools` → `Add-ons` zu öffnen und im Karteireiter `Get Add-ons` alle darin aufgelisteten Erweiterungen zu durchsuchen. Heruntergeladene Erweiterungen, die dem Anwender bereits als `.xpi` Datei zur Verfügung stehen, können ohne erneutes Herunterladen mittels Drag & Drop in das `Add-ons` Fenster installiert werden. Das Aufstellen einer „offiziellen“ Liste emp-

fohlener Erweiterungen[a] sollte nicht mit Zensur verwechselt werden. Für neue Erweiterungen, die noch keiner ausreichenden Prüfung unterzogen worden sind, wurde von Mozilla ein Entwicklerbereich mit dem Namen „Sandbox“[b] eingerichtet. Die Entwickler von Thunderbird versuchen auf diesem Weg die Entwickler von Erweiterungen dazu anzuhalten, ausschließlich gut programmierte und getestete Erweiterungen zu veröffentlichen. Erhält eine Erweiterung, die sich in der Sandbox befindet, ausreichend positive Beurteilungen, wird diese einem genauen Review unterzogen und bei Bestehen in die „offizielle“ Liste empfohlener Erweiterungen aufgenommen.

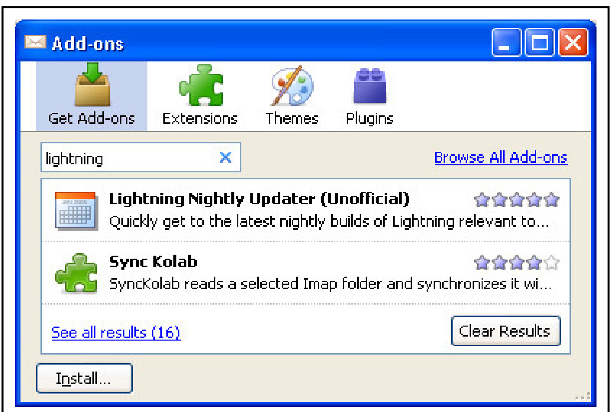

Abbildung 2.2: Add-ons Fenster

Als Mailclient stellt Thunderbird eine wichtige Schnittstelle zum Internet dar und ist als solche einer Vielzahl von Gefahren ausgesetzt. Leider ist es gängige Praxis, Nachrichten vom Mailserver des Providers abzurufen und diese ohne vorherige Virenprüfung im Posteingang abzulegen. Da es bei dieser Vorgehensweise einfach ist, Sicherheitslücken eines Mailclients auszunutzen, ist es wichtig, diesen kontinuierlich weiterzuentwickeln und Schwachstellen so früh wie möglich zu schließen. Zu diesem Zweck ist Thunderbird mit einem automatischen Aktualisierungsmechanismus, ähnlich dem Windows Update, ausgestattet. Bei jedem Start von Thunderbird wird überprüft, ob dieser oder eine installierte Erweiterung in einer neueren Version verfügbar ist. Um dies zu ermöglichen, wird für jede Erweiterung eine URL angegeben, von welcher die aktuellste Version abgerufen werden kann. Dadurch ist es auf einfache und schnelle Weise möglich, Updates, ausgehend von einem zentralen Punkt, allen weltweit verteilten Systemen zugänglich zu machen. Seit Thunderbird 3.0a1pre können Erweiterungen und deren Updates digital signiert werden. Auf diese Weise ist sichergestellt, dass keine Schadsoftware über ein gefälschtes Update installiert werden kann.

[a] https://addons.mozilla.org/en-US/thunderbird
[b] https://addons.mozilla.org/en-US/thunderbird/pages/sandbox

2.3 Bestehende Lösungen

Alle bisher für Thunderbird entwickelten Erweiterungen verfolgen das Ziel, Thunderbird durch Hinzufügen einer bestimmten Funktionalität zu erweitern oder dessen Aussehen zu verändern. Der Autor wird im Anschluss einen kurzen Einblick in die unter [a] geführten Erweiterungen geben, die mit Hilfe des unter dem Namen JackF entwickelten Frameworks hätten realisiert werden können.

Display Mail User Agent:[b] ermöglicht es, bei den Kopfdaten einer Nachricht ein Bild einzublenden, das anzeigt, mit welchem Mailclient die Nachricht versandt wurde. Dies ist möglich, da der Mailclient des Absenders üblicherweise in der versandten Nachricht eingetragen ist. Die unten dargestellte Abbildung zeigt, wie die Erweiterung eine mit Microsoft Outlook versandte Nachricht anzeigen würde.

Abbildung 2.3: Display Mail User Agent Beispiel

ConfigDate:[c] ermöglicht das Anpassen des Formats der Datumsspalte in der Nachrichtenübersicht. Thunderbird verwendet standardmäßig die vom Absender in einer Nachricht eingetragene und unter Umständen vom Absender gefälschte Absendezeit. Leider verabsäumte der Entwickler, eine Möglichkeit zu schaffen, anstatt der vom Absender in der Nachricht eingetragenen Information den Zeitpunkt des vom Empfängermailserver eingetragenen Empfangszeitpunktes, zu verwenden. Grundsätzlich wäre dies möglich, da jeder Mailserver die Uhrzeit, zu welcher er eine Nachricht empfängt, in die Nachricht einträgt, wodurch gefälschte Absendezeitpunkte erkannt werden könnten.

[a] https://addons.mozilla.org/en-US/thunderbird/browse/type:1/cat:all
[b] https://addons.mozilla.org/en-US/thunderbird/addon/562
[c] https://addons.mozilla.org/en-US/thunderbird/addon/901

Country Lookup:[a] stellt anhand der IP Adresse des Absenders einer Nachricht fest, in welchem Land sich dieser befindet. Anschließend wird bei den Kopfdaten der Nachricht eine kleine Länderflagge angezeigt.

Abbildung 2.4: Country Lookup Beispiel

IranZilla:[b] ähnlich wie ConfigDate ermöglicht IranZilla das Anzeigen einer Datumsspalte in iranischem Format.

Sender		Size	Date	Iranian Date
	•	2KB	30.08.2007 00:35	۸ شهریور ۱۳۸۶، ۰:۲۵
	•	3KB	04.09.2007 16:19	۱۳ شهریور ۱۳۸۶، ۱۶:۱۹
	•	2KB	05.09.2007 09:02	۱۴ شهریور ۱۳۸۶، ۹:۲
	•	1KB	05.09.2007 10:07	۱۴ شهریور ۱۳۸۶، ۱۰:۷
	•	4KB	05.09.2007 19:00	۱۴ شهریور ۱۳۸۶، ۱۹:۰
	•	104KB	06.09.2007 04:47	۱۵ شهریور ۱۳۸۶، ۴:۴۷
	•	2KB	06.09.2007 06:59	۱۵ شهریور ۱۳۸۶، ۶:۵۹
	•	5KB	06.09.2007 09:20	۱۵ شهریور ۱۳۸۶، ۹:۲۰
	•	3KB	06.09.2007 17:50	۱۵ شهریور ۱۳۸۶، ۱۷:۵۰
	•	3KB	06.09.2007 18:59	۱۵ شهریور ۱۳۸۶، ۱۸:۵۹
	•	5KB	07.09.2007 06:54	۱۶ شهریور ۱۳۸۶، ۶:۵۴
	•	2KB	07.09.2007 07:43	۱۶ شهریور ۱۳۸۶، ۷:۴۳

Abbildung 2.5: IranZilla Beispiel

MailClassifier:[c] benutzt einen bayesschen Filter, um Nachrichten zu kategorisieren. Die berechnete Kategorie wird danach in einer zusätzlichen Spalte in der Nachrichtenübersicht angezeigt und soll den Anwendern helfen, diese im Anschluss im richtigen Ordner abzulegen.

[a] https://addons.mozilla.org/en-US/thunderbird/addon/3595
[b] https://addons.mozilla.org/en-US/thunderbird/addon/4976
[c] https://addons.mozilla.org/en-US/thunderbird/addon/3376

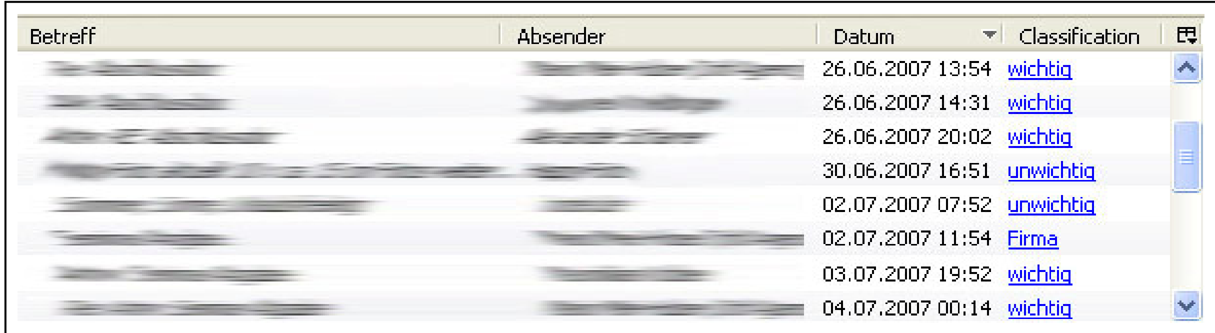

Abbildung 2.6: MailClassifier Beispiel

Sender Verification Anti-Phishing Extension:[a] versucht, mit Hilfe von SPF (Sender Policy Framework) gefälschte Emailabsender zu erkennen. SPF ist eine Erweiterung von SMTP und gibt Domaininhabern die Möglichkeit anzuführen, welche Systeme berechtigt sind, Nachrichten für diese zu verschicken. Wird eine vermeintlich gefälschte Nachricht erkannt, ist die Gefahr hoch, dass es sich um Spam oder um einen sogenannten „Phishingangriff" handelt, und es wird eine Warnung in den Kopfdaten der Nachricht angezeigt. Der Begriff Phishing wird verwendet, um zu beschreiben, dass jemand versucht, auf kriminelle oder betrügerische Weise sensible Daten, wie zum Beispiel Benutzernamen, Sozialversicherungsnummern oder auch Kreditkarteninformationen, zu beschaffen.

Abbildung 2.7: Sender Verification Anti-Phishing Extension

Show InOut:[b] fügt zwei neue Spalten in die Nachrichtenübersicht ein. Die erste Spalte zeigt an, ob die Nachricht gesendet oder empfangen wurde. Die zweite Spalte ergänzt die Übersicht um die Empfängeremailadressen für gesendete und Absenderemailadressen für empfangene Nachrichten. Dies ist besonders praktisch für Anwender, die gesendete und empfangene Nachrichten im selben Ordner aufbewahren.

[a] https://addons.mozilla.org/en-US/thunderbird/addon/345
[b] https://addons.mozilla.org/en-US/thunderbird/addon/3492

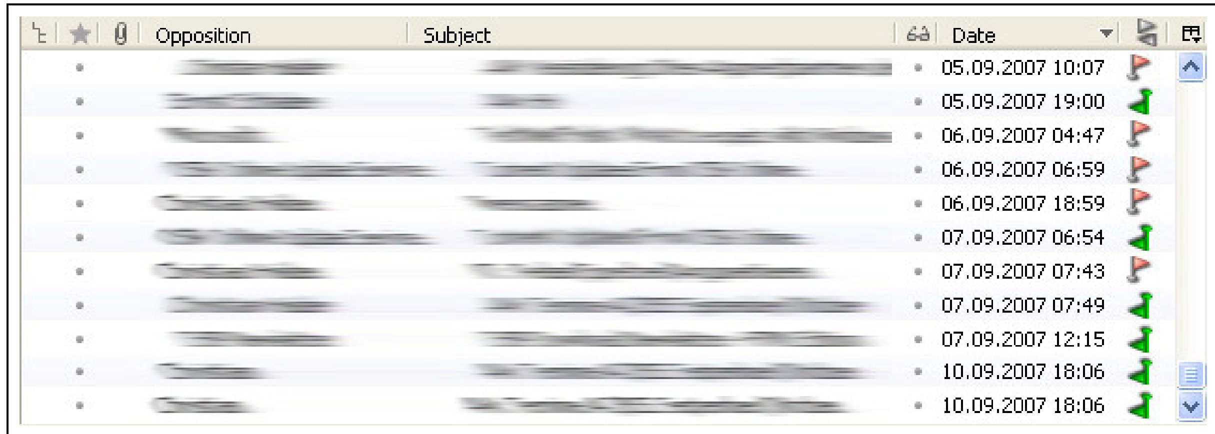

Abbildung 2.8: Show InOut Beispiel

SenderFace:[a] ist eine Erweiterung, die es ermöglicht, eine Emailadresse mit einem Bild des Absenders zu verknüpfen. Dieses Bild wird in den Kopfdaten einer Nachricht angezeigt.

Abbildung 2.9: SenderFace Beispiel

Message Level Authentication:[b] versucht Vorhersagen zu treffen, ob es sich bei einer Nachricht um eine Spam-, Phishing- oder um eine „wirkliche" Nachricht handelt. Dazu werden Informationen, die eine Vorhersage erlauben, über einen Webservice von PhishTank beschafft. Das Ergebnis wird in den Kopfdaten einer Nachricht angezeigt.

Вахраненко	Re: Автоакустика Soundstream, Alpine, ...	25.03.2008 22:44
Terrie Blevins	**don't be shy**	**25.03.2008 22:40**
Дрогаченко	FW: Пошив одежды.	25.03.2008 22:37
Charity	herk	25.03.2008 22:32
Denice Geese	skittles	25.03.2008 21:31
humphrey omead	Low prices everyday for top quality brand...	25.03.2008 20:05
Tami Cummins	Swagger rep1!cated brands	25.03.2008 19:05

Authentication Result: Bad From Address (details) Try Again Sender Report
Reputation Result: Beware of this Sender

Abbildung 2.10: Message Level Authentication Beispiel

[a] https://addons.mozilla.org/en-US/thunderbird/addon/2843
[b] https://addons.mozilla.org/en-US/thunderbird/addon/2260

Spamness:[a] ist eine derzeit noch in der Sandbox befindliche Erweiterung. Einige Mailserver fügen in den Header einer Nachricht einen Wert für die Wahrscheinlichkeit, dass es sich um eine Spamnachricht handelt, ein. Mit dieser Erweiterung ist es möglich diesen Wert grafisch in einer Spalte in der Nachrichtenübersicht anzuzeigen.

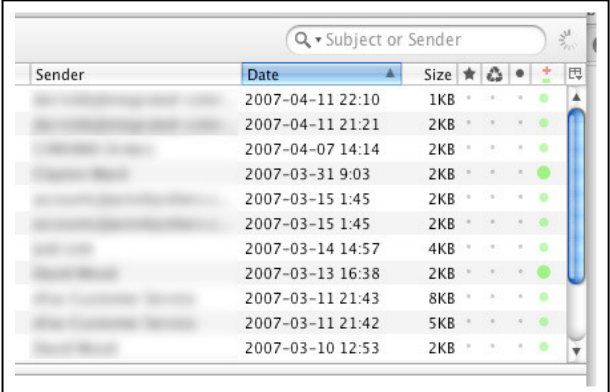

Abbildung 2.11: Spamness Beispiel

Enigmail:[b] erlaubt es, Nachrichten mittels RSA sowohl zu verschlüsseln, als auch zu signieren. Damit stellt es eine sichere Kommunikationsmöglichkeit zwischen zwei Endpunkten zur Verfügung. Auch wenn es nicht möglich wäre, Enigmail mit JackF zu realisieren, so ist diese Erweiterung auf Grund ihrer Bedeutung eine Erwähnung wert.

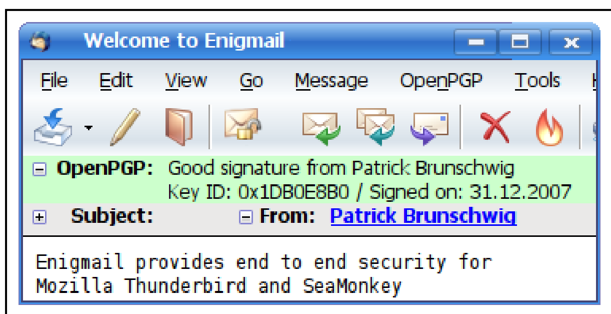

Abbildung 2.12: Enigmail Beispiel

2.4 Grenzen der bestehenden Lösungen

Alle oben genannten Erweiterungen stellen Zusatzfunktionen in Thunderbird zur Verfügung und sind für bestimmte Benutzergruppen wertvoll. Der Versuch, ein Framework zu gestalten, welches eine Schnittstelle für andere Erweiterungen bietet und deren Einbindung erleichtert, wurde noch nicht unternommen. Durch die Verwendung eines solchen Frameworks könnte der Gesamtnutzen der PlugIn-Technologie, die Thunderbird nutzt, gesteigert werden. Ein sol-

[a] https://addons.mozilla.org/en-US/thunderbird/addon/4798
[b] https://addons.mozilla.org/en-US/thunderbird/addon/71

ches Framework muss eine stabile und flexible Schnittstelle bieten, wodurch Erweiterungen auch dann noch weiter benutzt werden können, wenn an Thunderbird Änderungen vorgenommen werden. Momentan sind sieben der neun vorgestellten Erweiterungen noch nicht für den sich in der Beta Phase befindlichen Thunderbird 3 geeignet. Es bleibt offen, ob diese mit dem Erscheinen von Thunderbird 3 einsatzfähig sein werden.

Bei über 62 Millionen Downloads ist es verwunderlich, dass auf der offiziellen Webseite für Thunderbirderweiterungen[a] zum 04. März 2008 lediglich 364 empfohlene und weitere 119 in der Sandbox befindliche Erweiterungen, gelistet sind. Ein Grund dafür könnte sein, dass es für Entwickler sehr zeitintensiv ist, sich in die Programmierung einer Erweiterung einzuarbeiten. Ein Programmierer, der lediglich die Programmiersprache JS beherrscht, könnte alle genannten Erweiterungen, mit der Ausnahme von Enigmail, mit der Hilfe von JackF implementieren, ohne dass er sich Kenntnisse über XUL, XPCOM und Ähnlichem aneignen müsste.

Die meisten Erweiterungen verwenden den einfachsten Weg gewünschte Informationen anzuzeigen und fügen dazu zum Beispiel in der Nachrichtenübersicht eine Spalte ein. Diese Spalte wird von Thunderbird häufig aktualisiert, so etwa auch beim Bewegen der Maus über eine Zeile in der Nachrichtenübersicht. Auch wenn Thunderbird mehrere Threads benutzt, um Aufgaben wie den Mailversand und -empfang im Hintergrund abzuarbeiten, werden Änderungen an der Benutzeroberfläche immer vom sogenannten `MainThread` durchgeführt. Dies ist notwendig, da die Elemente der Benutzeroberfläche nicht von mehreren Threads aus verwendet werden dürfen. Eine Konsequenz davon ist, dass die Funktionen der Erweiterung, die für das Berechnen des Zelleninhaltes zuständig sind, beim Aktualisieren der Benutzeroberfläche auch im `MainThread` ausgeführt werden. Dies impliziert einen entscheidenden Nachteil für die Benutzeroberfläche, da diese während der Anfrage jeder einzelnen Zelle, der Spalte in der Nachrichtenübersicht für den Zeitraum, in dem die Auswertung geschieht und die darzustellende Information berechnet wird, nicht auf Benutzereingaben reagieren kann. Unter der Annahme, dass eine Erweiterung sich eines Webservices bedient, und dieser ungefähr 100 ms für eine Antwort benötigt, wäre beim Anzeigen von 20 Zellen die Benutzeroberfläche mindestens 2 Sekunden lang eingefroren.

Da auch Thunderbird selbst dieses Problem hätte, wenn dieser ständig alle in der Nachrichtenübersicht dargestellten Informationen erneut aus den gespeicherten Nachrichten extrahie-

[a] https://addons.mozilla.org/en-US/thunderbird

ren müsste, erzeugt dieser für jeden Nachrichtenordner eine kleine .msf Datei, in welcher er einen Index für die am häufigsten verwendeten Informationen ablegt. Zu diesen zählen zum Beispiel, ob die Nachricht bereits gelesen wurde, der Absender, der Empfänger, der Betreff, und andere mehr. Alle Informationen, die aus diesem Index gelesen werden können, sind für Thunderbird schneller zugänglich, als wenn er diese erst aus der Originalnachricht laden müsste. Diesen Mechanismus benutzen auch manche Erweiterungen und speichern ihre zusätzlichen Informationen in diesem Index ab. Dabei handelt es sich um eine eher fragwürdige Art, Ergebnisse für ein erneutes Anzeigen zwischenzuspeichern, denn wenn eine Erweiterung nicht länger benötigt und aus Thunderbird deinstalliert wird, belässt diese die eingefügten Informationen in der .msf Datei. Auch wenn die .msf Datei aus allen in einem Ordner gespeicherten Nachrichten neu erzeugt werden kann, ist dennoch keine klare Trennung zwischen den originalen, unveränderten Daten und den, von Erweiterungen erzeugten Inhalten, möglich. Um sich den zusätzlichen Mehraufwand, den das Abspeichern von Ergebnissen mit sich bringt, zu sparen, wird bei einigen Erweiterungen gänzlich auf diese Möglichkeit verzichtet. Stattdessen wird auf Kosten der Geschwindigkeit jedes Mal, wenn eine Zelle aktualisiert wird, die benötigte Information von Neuem berechnet.

2.5 Lösungsansatz

Ein anzustrebendes Ziel ist es, die Anzahl von für Thunderbird erhältlichen Erweiterungen im Sicherheitskontext zu erhöhen. Ein Vorschlag wäre die Schaffung eines Forums, in welchem Ideen für brauchbare Erweiterungen von unerfahrenen Anwendern ohne Programmierkenntnisse veröffentlicht werden könnten, die dann von erfahrenen Entwicklern aufgegriffen und in die Realität umgesetzt werden könnten. Auch wenn ein derartiges Forum ein erster Schritt in die richtige Richtung wäre, und die Anzahl der zur Verfügung stehenden Erweiterungen damit erhöht werden könnte, wäre das Grundproblem, nämlich dass das Entwickeln von Erweiterungen für Thunderbird einen hohen Einarbeitungsaufwand bedeutet, damit nicht gelöst.

Zielführender als ein Forum wäre es, Probleme, mit denen sich Programmierer herkömmlicher Erweiterungen auseinandersetzen müssen, auf einer abstrakteren Ebene zu lösen und somit den Entwicklungsprozess zu vereinfachen. In diesem Zusammenhang ist es erwähnenswert, dass viele der beliebtesten Erweiterungen keine Veränderungen an der Funktionalität von Thunderbird vornehmen, sondern lediglich zusätzliche Informationen über Nachrichten anzeigen. Gerade diese Erweiterungen könnten meist in wenigen Codezeilen abgehandelt

werden. So ist etwa die in Spamness[a] für die Aufbereitung der grafischen Symbole verantwortliche Funktion 22 Zeilen lang. Ein Lösungsansatz auf abstrakter Ebene hätte den Vorteil, dass Erweiterungen schneller realisiert werden könnten und eine breitere Entwicklergemeinde angesprochen werden würde.

Um dieses Vorhaben in die Realität umzusetzen zu können, ist ein Framework zu entwickeln, welches Erweiterungen in Thunderbird integriert und diesen erlaubt, Nachrichten auf einfache und strukturierte Weise bezüglich bestimmter Kriterien zu prüfen, und das Ergebnis dieser Prüfung auszugeben. Um eine klare Trennung zwischen Erweiterungen, die das Framework benutzen, und „herkömmlichen" Erweiterungen zu ermöglichen, werden erstere in Folge als „Pluggies" bezeichnet. Welche Art von Prüfung ein Pluggy durchführt, soll diesem selbst überlassen werden. Es wird eine Bandbreite beginnend bei einfachen Aufgaben, wie dem Zählen aller Empfänger einer Nachricht, bis hin zu komplexeren Aufgaben, wie dem Berechnen der Spamwahrscheinlichkeit einer Nachricht, abgedeckt. Daher wird diese Prüfung folglich mit dem treffenderen Begriff „Analyse" betitelt. Die Kommunikation zwischen Thunderbird und dem eingebundenen Pluggy darf ausschließlich über das Framework erfolgen. Einige dazu vom Framework zu übernehmende Aufgaben wurden bereits unter *2.4 Grenzen der bestehenden Lösungen* erwähnt und werden nachfolgend kurz erläutert:

- *Bereitstellen einer einfachen Schnittstelle:* Zwischen dem Framework und einem Pluggy muss eine einfache Schnittstellte bestehen, welche mit wenigen Funktionen erlaubt, wahlweise bis zu 3 verschiedene Ausgabeformate für das Ergebnis einer Analyse zu wählen. Das Framework unterstützt die Ausgabe der Analyseergebnisse in grafischer (✘, ○, ✔), numerischer (1, 2, …) und alphanumerischer (hoch, mittel, niedrig, …) Form.
- *Multithreading:* Das Abarbeiten von Analysen kann unter Umständen eine ressourcenintensive Aufgabe sein, die viel Zeit in Anspruch nimmt. Damit die Benutzeroberfläche während einer Analyse auf Benutzereingaben reagieren kann, muss das Framework die Analyse durch im Hintergrund arbeitenden Threads bewerkstelligen. Die Verwaltung dieser Threads muss für die Pluggies transparent erfolgen.
- *Cachen von Analyseergebnissen:* Die Analyseergebnisse müssen vom Framework zwischengespeichert werden, um bei wiederholtem Bedarf, ohne erneute Berechnung verfügbar zu sein. Der Zeitraum, in dem ein zwischengespeichertes Analyseergebnis

[a] https://addons.mozilla.org/en-US/thunderbird/addon/4798

gültig ist, muss von der Erweiterung frei bestimmt werden können, da diese Entscheidung nicht pauschal für alle Analyseergebnisse getroffen werden kann. So ist etwa die Anzahl der Empfänger fortwährend gültig und würde sich bei erneutem Analysieren nicht ändern. Im Gegensatz dazu ist die Spamwahrscheinlichkeit durch Hinzufügen neuer Bedingungen durchaus veränderbar. Die zwischengespeicherten Analyseergebnisse sind getrennt von den Nachrichten zu speichern. Dies hat den Vorteil, dass der interne Zustand von Thunderbird unberührt belassen wird, und das Framework von zukünftigen Änderungen in dessen Datenhaltung unabhängig ist.

- *„Gewöhnliche" Installation:* Das Installieren eines Pluggies darf sich vom Installieren herkömmlicher Erweiterungen nicht unterscheiden. Damit wird ausgeschlossen, dass unerfahrene Anwender daran scheitern, zusätzliche Pluggies zu verwenden.
- *Automatische Updates:* Der automatische Updatemechanismus von Thunderbird muss sowohl für das Framework, als auch für alle installierten Pluggies funktionieren. Dadurch ist es den Entwicklern möglich, Änderungen am Framework oder an Pluggies allen Anwendern auf einfache Weise zugänglich zu machen. Mit der Erfüllung der schon zuvor erwähnten Forderung, dass Pluggies wie herkömmliche Erweiterungen installiert werden können sollen, ist der automatische Updatemechanismus für diese uneingeschränkt verwendbar.

3 Erweiterungen von Thunderbird

Das folgende Kapitel soll einen Einblick über die Programmierung von Erweiterungen für Thunderbird geben. Da Thunderbird, Firefox und andere aus dem Mozilla Projekt entstandene Anwendungen auf derselben Codebasis beruhen, sind die in diesem Kapitel vermittelten Techniken zum Großteil auf die gesamte Produktfamilie anwendbar. Auch wenn kein Spezialwissen über C++, JS oder das Gestalten von Webseiten vorausgesetzt wird, so sind Grundkenntnisse in diesen Bereichen von Vorteil. Es wird nicht versucht, eine vollständige Aufzählung aller in HTML, JS, CSS, und anderen mehr benutzten Elemente oder Techniken zu geben, es soll vielmehr deren Verwendung in Thunderbird erläutert werden. Für den interessierten Leser, der mehr über die mit Thunderbird in Verbindung stehenden Technologien erfahren möchte, würde der Autor gerne auf die ihm sehr hilfreich gewesenen Quellen verweisen. Diese sind:

Anleitungen zum Schreiben einer Erweiterung:

How to create Firefox extensions
http://roachfiend.com/archives/2004/12/08/how-to-create-firefox-extensions/

Getting started with extension development
http://kb.mozillazine.org/Getting_started_with_extension_development

Creating Applications with Mozilla
http://www.csie.ntu.edu.tw/~piaip/docs/CreateMozApp/

The XPToolkit Architecture
http://www.mozilla.org/xpfe/xptoolkit/

Firefox Extension Development Tutorial: Overview
http://www.rietta.com/firefox/Tutorial/overview.html

Java Script:

Core JavaScript 1.5 Guide
http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

JavaScript Operators
http://www.w3schools.com/js/js_operators.asp

JavaScript - Mozilla Developer Center
http://developer.mozilla.org/en/docs/JavaScript

A re-introduction to JavaScript – Mozilla Developer Center
http://developer.mozilla.org/en/docs/A_re-introduction_to_JavaScript

This is the JavaScript reference
http://www.topxml.com/javascript/default.asp

JavaScript
http://www.kevlindev.com/tutorials/javascript/inheritance/

DOM:

Document Object Model (DOM) Level 2 Core
http://www.w3.org/TR/DOM-Level-2-Core/Overview.html

XPCOM:

Creating XPCOM Components
http://www.mozilla.org/projects/xpcom/book/cxc/html/newbookTOC.html

Implementing XPCOM components in JavaScript
http://kb.mozillazine.org/Implementing_XPCOM_components_in_JavaScript
XPCOM API Reference – Mozilla Developer Center
http://developer.mozilla.org/en/docs/XPCOM_API_Reference
XPCOM Interfaces
http://www.xulplanet.com/references/xpcomref/
XPCOM Part 1-5: An introduction to XPCOM (IBM)
http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html
http://www-128.ibm.com/developerworks/webservices/library/co-xpcom2.html
http://www-128.ibm.com/developerworks/webservices/library/co-xpcom3.html
http://www-128.ibm.com/developerworks/webservices/library/co-xpcom4/
http://www-128.ibm.com/developerworks/webservices/library/co-xpcom5.html

XPIDL:
XPIDL – Mozilla Developer Center
http://developer.mozilla.org/en/docs/XPIDL:xpidl
Frequently Asked Questions about XPConnect and XPIDL
http://www.mozilla.org/scriptable/faq.html

XUL:
XUL Tutorial
http://www.xulplanet.com/tutorials/xultu/
XUL Element and Script Reference
http://www.xulplanet.com/references/elemref/quickref.html
XUL Element Reference
http://www.xulplanet.com/references/elemref/

More Resources:
Extension development
http://kb.mozillazine.org/Extension_development

3.1 Einführung Mozilla Programmierung

Erweiterungen werden für Thunderbird und andere dem Mozilla Projekt abstammende Anwendungen in Dateien mit der Dateiendung `.xpi` bereitgestellt. Diese `.xpi` Dateien sind komprimierte Dateien, die sowohl das Aussehen, als auch die Funktionalität einer Erweiterung beinhalten. Die Ordnerstruktur einer `.xpi` Datei ist vorgegeben und muss eingehalten werden.

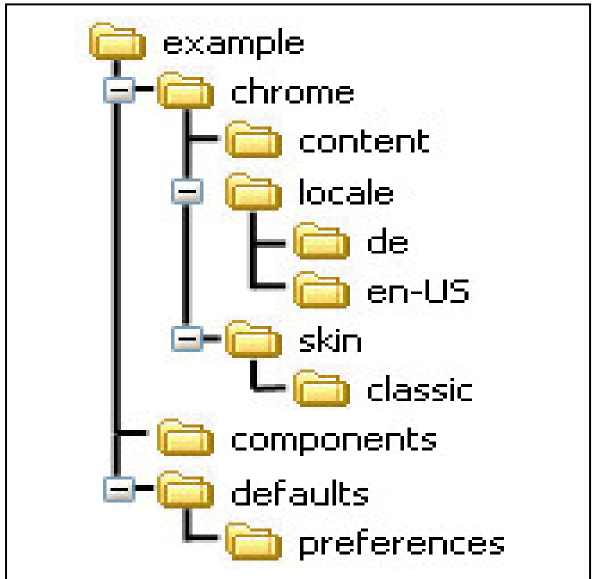

Abbildung 3.1: Struktur einer .xpi Datei

Kurz umrissen, erfüllen die oben abgebildeten Ordner folgende Funktionen:

- *chrome:* Dieser Ordner nimmt in der `.xpi` Datei eine Sonderstellung ein. Er wird 2-fach komprimiert. Eine ausführlichere Erklärung dazu wird in Kapitel *3.1.8 Packaging* gegeben.
- *content:* Die Benutzeroberfläche einer Erweiterung wird in diesem Ordner abgelegt.
- *defaults:* In diesem Ordner sind Standardwerte, die beim ersten Starten der Erweiterung benutzt werden, abgelegt.
- *locale:* Der Inhalt dieses Ordners sind „Stringtabellen“, welche eine Übersetzung in verschiedene Sprachen erlauben, ohne die Erweiterung zu verändern.
- *skin:* Ähnlich wie bei Webseiten können Erweiterungen mit Vorlagen in deren Aussehen verändert werden. Diese Vorlagen befinden sich im `skin` Ordner.
- *components:* Der components Ordner beinhaltet sogenannte XPCOM Components, in welchen größere Programmteile einer Erweiterung realisiert werden [TuOe03].

Der Inhalt der Ordner muss Thunderbird mitgeteilt werden. Dies kann auf zwei Arten erfolgen. Zum einen können in den Ordnern `content`, `locale` und `skin` Dateien mit dem Namen `contents.rdf` abgelegt werden, welche den Inhalt der Ordner näher beschreiben. Zum anderen ist es seit Thunderbird 1.5 möglich, den gesamten Inhalt aller Ordner und Unterordner in einer Datei mit dem Namen `chrome.manifest` zu beschreiben. Diese Variante ist die einfachere, zu bevorzugende und wird in Kapitel *3.1.7 Install.rdf und chrome.manifest* be-

schrieben. Beim Inhalt einer RDF Datei handelt es sich um Informationen, die mit Hilfe einer Beschreibungssprache, die mit Subjekt-Prädikat-Objekt Regeln Informationen beschreibt, abgelegt werden [Cha01]. Um das Programmieren einer Erweiterung besser veranschaulichen zu können, wird schrittweise eine kleine Erweiterung aufgebaut, die einen zusätzlichen Button in die Menüleiste einfügt, welcher, wenn er angeklickt wird, ein Dialogfenster öffnet. Es werden immer nur die Teile des Quellcodes aufgeführt, die zum Verständnis beitragen. Für den gesamten Quellcode wird auf Kapitel *8.1 Quellcode Example* verwiesen.

3.1.1 XUL (XML User Interface Language)

XUL ist eine vom XML 1.0 Standard abgeleitete Markup Language, die zum Gestalten von Benutzeroberflächen entwickelt wurde. Ähnlich DHTML erlaubt sie das dynamische Gestalten von Webseiten. Dies ermöglicht XUL, auf Benutzereingaben zu reagieren und wenn nötigt, Änderungen an der Darstellung durchzuführen. XUL erlaubt zusätzlich einige Features, die in HTML 4.0 eingeführt wurden. So kann eine in XUL gestaltete Benutzeroberfläche mittels CSS (Cascading Style Sheets) verändert oder mittels JS um zusätzliche Funktionalität erweitert werden. Die Verwendung von XUL ist jener von HTML sehr ähnlich, wie das folgende Beispiel, das den möglichen Inhalt einer XUL Datei zeigt, veranschaulicht.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
    title="Example" class="dialog">
  <groupbox align="center" orient="horizontal">
    <vbox>
      <text value="Example" style="font-weight: bold; font-
      size: x-large;"/>
      <text value="ver. 1.0"/>
      <separator class="thin"/>
      <hbox>
        <text value="created by:" style="font-weight: bold;"/>
        <text value="Christian Haider"/>
      </hbox>
      <hbox>
        <label value="your name:"/>
        <textbox id="name" preftype="char" flex="1" value="Christian"/>
      </hbox>
      <spacer flex="1"/>
      <button label="Close" accesskey="c" oncommand="window.close();"/>
    </vbox>
  </groupbox>
</window>
```

Abbildung 3.6: XUL Beispiel Quellcode

Da es sich bei .xul um eine XML Datei handelt, beginnt der Inhalt, wie bei XML üblich, mit einem Verweis auf die XML Version. Die zweite Zeile des Beispiels verweist auf die bei XUL standardmäßig verwendeten CSS Dateien, welche immer importiert werden sollten. Es können beliebig viele CSS Dateien benutzt werden, wobei `chrome://global/skin/` immer an erster Stelle stehen sollte.

Der Aufbau des restlichen Dokumentes ist dem einer normalen HTML Seite ähnlich. Die folgende Abbildung zeigt, wie das obige Beispiel von der Präsentationsschicht dargestellt wird.

Abbildung 3.2: XUL Beispiel Ausgabe

Die Benutzeroberfläche von Thunderbird ist gänzlich in XUL gestaltet worden, wodurch die eigentliche Anwendungslogik von der Benutzeroberfläche getrennt wird. Zur Darstellung der Benutzeroberfläche wird eine Layout-Engine namens Gecko verwendet. Diese baut aus den in XUL vorliegenden Objekten eine Baumstruktur, basierend auf Knoten und Kanten, auf. Das durch diese Vorgehensweise erzeugte Modell, wird allgemein als DOM (Document Object Model) bezeichnet. Aus dem oben angeführten Beispiel würde Gecko einen Baum mit folgendem Inhalt erzeugen.

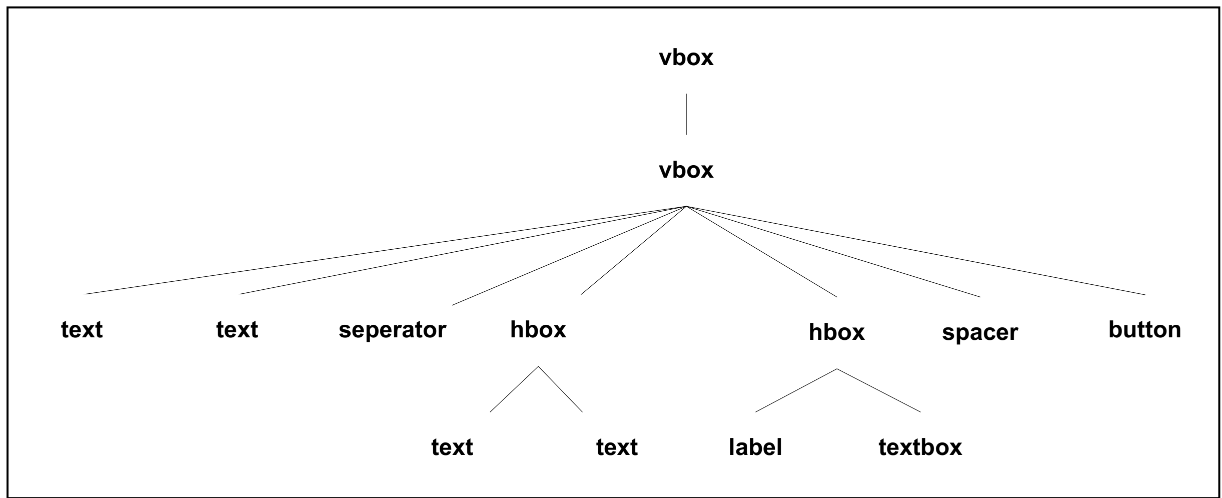

Abbildung 3.3: XUL Beispiel als DOM

Die für die Benutzeroberfläche benötigten `.xul` und `.js` Dateien werden im `content` Ordner abgelegt. Die für das Ändern des Aussehens verwendeten `.css` Dateien befinden sich üblicherweise im `skin` Ordner. Es wäre auch möglich alle Dateien gemeinsam zu speichern, jedoch ist es durch eine Trennung später möglich, mit einem Skin Einfluss auf das Aussehen zu nehmen. Näheres siehe dazu in Kapitel *3.1.4 Skins, Styles and Themes*.

3.1.2 CHROME

Da HTML und XUL Dateien von Gecko gleich gehandhabt werden, wäre es ohne geeignete Schutzmechanismen möglich, eine Nachricht dahingehend zu gestalten, dass diese schadhaften Code in Thunderbird einschleust, und Thunderbird diesen ausführt. Aus diesem Grund werden alle in Thunderbird installierten Erweiterungen und deren Dateien in einem speziellen virtuellen Verzeichnis, dem CHROME, abgelegt und erhalten intern eine URL, die anstatt mit `http://` oder `file://`, mit `chrome://` beginnt. Auf diese Weise kann zwischen extern abgerufenen Inhalten und Inhalten mit speziellen Berechtigungen unterschieden werden. Externe Inhalte werden in ihren Rechten aus Sicherheitsgründen stark eingeschränkt und können zum Beispiel nicht auf das lokale Dateisystem zugreifen. Wie bereits erwähnt erhalten im CHROME abgelegte Dateien besondere Rechte und können so zum Beispiel die Benutzereinstellungen verändern, auf Nachrichten zugreifen oder Peripheriegeräte verwenden. Das Registrieren einer Datei als im CHORME befindlich und das damit verknüpfte Zuweisen einer Chrome URL übernimmt Thunderbird beim Installieren einer Erweiterung und muss nicht explizit vom Anwender durchgeführt werden.

Thunderbirds Benutzeroberfläche besteht aus einer Ansammlung von vielen XUL Dateien, die alle im CHROME registriert sind. Zwar sind diese Dateien größer und aufwendiger als jene von Erweiterungen, das Prinzip ist dennoch dasselbe. Dies vor Augen ist es leicht vorstellbar, dass eine Erweiterung das Aussehen oder die Funktionalität von Thunderbird verändern kann. Für den Anwender mag es den Anschein haben, dass Thunderbird verändert wurde, tatsächlich bleibt dieser, in Bezug auf den Code, unverändert. Die Layout-Engine hat lediglich beim Aufbauen des DOMs, die in XUL Form vorliegenden Elemente der Erweiterung mit den Elementen von Thunderbird durch einen Mechanismus namens Overlaying vereint.

3.1.3 Overlays

Overlays sind XUL Objekte, die bildlich gesehen auf eine Art Klarsichtfolie gezeichnet sind und an beliebiger Stelle auf die Benutzeroberfläche gelegt werden können, um diese genau an gewünschter Stelle um die aufgezeichneten Elemente zu ergänzen.

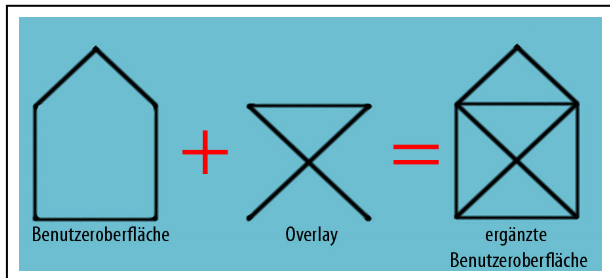

Abbildung 3.4: Overlay

Auf diese Weise erreicht man zwei Ziele. Zum einen sind Elementgruppen, die mehrfach benötigt werden, in einem Overlay realisierbar und können beliebig oft Anwendung finden. Ein gutes Beispiel dafür ist die Symbolleiste, welche sowohl im Hauptfenster zu sehen ist, als auch dann, wenn eine Nachricht geöffnet wird. Dies reduziert nicht nur den Entwicklungsaufwand, sondern erhöht auch die Wartbarkeit.

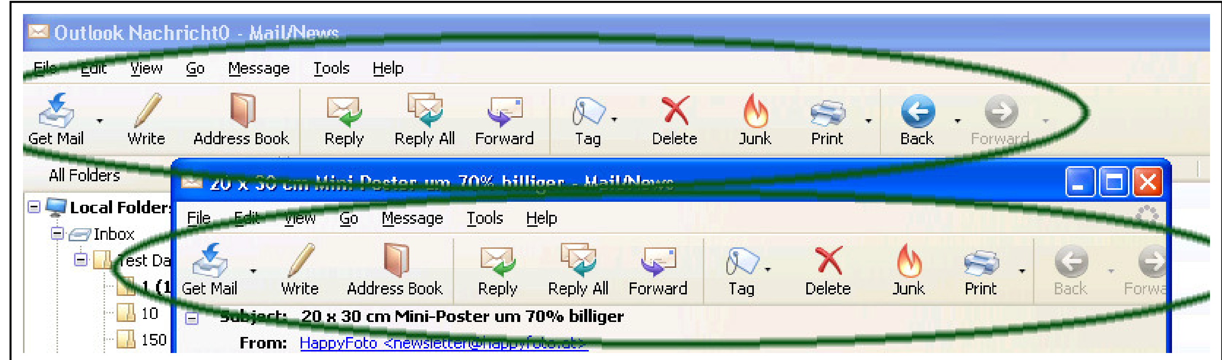

Abbildung 3.5: Menüleiste als Overlay

Zum anderen bietet der Einsatz von Overlays die Möglichkeit, die Benutzeroberfläche im Nachhinein zu erweitern, ohne dass ein erneutes Kompilieren der Anwendung erforderlich ist. Zu dessen Umsetzung ist die Verwendung eines Mechanismus notwendig, der nach Overlays sucht und diese berücksichtigt. Im Zuge dessen muss angegeben werden, welche Elemente durch ein Overlay erweitert werden sollen. Dies geschieht entweder durch die im Hauptverzeichnis einer Erweiterung abgelegte `chrome.manifest` Datei oder durch eine im `content` Ordner einer Erweiterung abgelegte `contents.rdf` Datei. Das unten angeführte Beispiel zeigt jenen Ausschnitt einer `contents.rdf` Datei, der für das Aufzählen der Overlays verantwortlich ist.

```
<RDF:Seq RDF:about="urn:mozilla:overlays">
  <RDF:li resource="chrome://messenger/content/messenger.xul"/>
</RDF:Seq>
<RDF:Seq about="chrome://messenger/content/messenger.xul">
  <RDF:li>chrome://jackf/content/overlay.xul</RDF:li>
</RDF:Seq>
```

Abbildung 3.6: contents.rdf Overlay Auflistung

Die zweite Zeile gibt an, dass Elemente der `messenger.xul` Datei mit einem Overlay ergänzt werden sollen. In der vierten Zeile wird beschrieben, dass eine Liste von Overlays folgt. Diese Liste besteht aus einem einzelnen Element, welches in der 5 Zeile geschrieben steht.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/global.css"
type="text/css"?>
<overlay id="jackfmenuebaroverlay"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <toolbar id="mail-bar2" >
    <toolbarbutton id="example" insertbefore="button-getmsg" oncommand="
      var win = window.openDialog
         ('chrome://example/content/example.xul','','chrome');
      win.focus ();">
      <image src="chrome://example/skin/icons.png"/>
      <text value="JackF"/>
    </toolbarbutton>
  </toolbar>
</overlay>
```

Abbildung 3.7: Beispiel Overlay Quellcode

Der Unterschied zwischen einer normalen `.xul` Datei, welche ein Dialogfenster beinhaltet, und einem Overlay besteht darin, dass wie im oben angeführten Beispiel das `window` Element durch ein `overlay` Element ersetzt wird. Beim Platzieren des Overlays sucht Gecko nach Elementen in der bestehenden Benutzeroberfläche, die die gleiche ID haben wie ein im `overlay` enthaltenes Element. Im Beispiel ist dies „`mail-bar2`“. Wird ein solches Element gefunden, fügt Gecko das Element des `overlays` an gewünschter Stelle ein. Für das Beispiel bedeutet dies, dass das Element `toolbar` mit der ID „`mail-bar2`“ mit dem Element `tool-barbutton` mit der ID „`jackfbutton`“ ergänzt wird. Im Falle, dass kein Element mit passender ID gefunden wird, wird dieser Teil des `overlays` ignoriert. Das für den `toolbar-button` verwendete Symbol ist im `skin` Ordner mit dem Dateinamen `jackf.png` abgelegt. Das obige Beispiel führt, wie in der unten dargestellten Abbildung ersichtlich, dazu, dass ein zusätzlicher Button in der Menüleiste eingefügt wird.

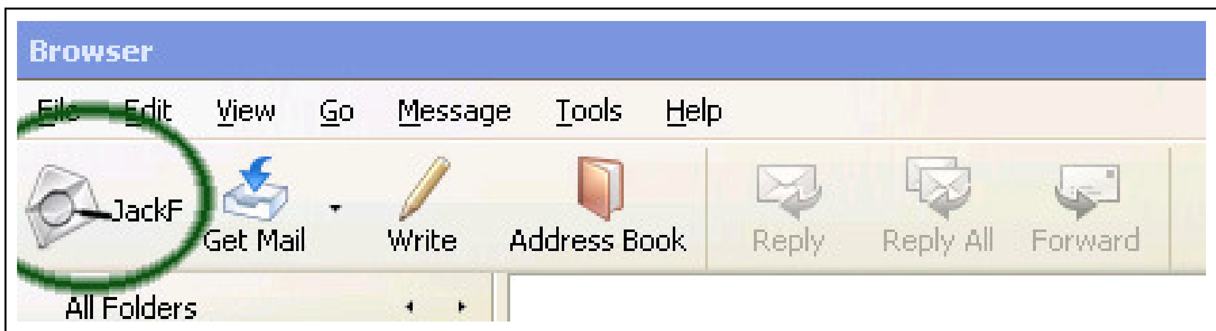

Abbildung 3.8: Beispiel Overlay Ausgabe

Um die ID der gewünschten Position zu finden, gibt es mehrere Möglichkeiten. Etwa könnte der Quellcode von Thunderbird betrachtet und die gewünschte Stelle gesucht werden. Dies

ist, bedingt durch die Länge des Quellcodes, mühsam, weshalb eine für Entwickler praktische Erweiterung namens DOM Inspector[a] gute Dienste leistet.

3.1.4 Skins, Styles and Themes

Wie bereits erwähnt, kann auf XUL Dateien mittels CSS (Cascading Style Sheets) Einfluss auf das Aussehen derer Darstellungen genommen werden. Dabei wird nur das Aussehen verändert, die Funktionalität bleibt dabei erhalten.

Sollte der Wunsch bestehen, Einfluss auf das gesamte Aussehen von Thunderbird zu nehmen, ist dies durch die Verwendung eines Skins möglich. Ein Skin kann wie eine Erweiterung installiert werden und beinhaltet zahlreiche `.css` Dateien. Das Erstellen dessen unterscheidet sich geringfügig von dem von Erweiterungen und wird nicht weiter behandelt. Eine brauchbare Anleitung dazu ist unter [b] abrufbar.

Da auch eine Erweiterung durch ein Skin verändert werden können soll, werden die `.css` Dateien üblicherweise nicht gemeinsam mit anderen Dateien einer Erweiterung vermischt abgelegt, sondern in einem Unterordner des `skin` Ordners, der mit dem Namen `classic` benannt wird. Entscheidend für die Namensgebung des Unterordners ist, dass der Standardskin von Mozilla den Namen `classic` trägt, und somit beim Installieren eines neuen Skins für Thunderbird automatisch auch die Erweiterung angepasst wird.

3.1.5 Localization

Da Erweiterungen in verschiedene Sprachen übersetzt werden, sollten die auf der Benutzeroberfläche ausgegebenen Texte nicht direkt in der `.xul` Datei gespeichert werden. Empfehlenswert ist es, alle Texte in einer Tabelle abzulegen und bei Bedarf anstatt des Textes einen Verweis auf die Zeile, in welcher der Text steht, zu verwenden. Auf diese Weise ist es möglich, die Stringtabelle zu veröffentlichen und Anwender zu bitten, diese von einer Sprache in eine andere zu übersetzen. Der Übersetzer braucht keine Programmierkenntnisse sondern er muss lediglich die Ursprungssprache und die Zielsprache kennen. Die Stringtabellen werden in Dateien mit der Dateiendung `.dtd` gespeichert. Um Thunderbird zu ermöglichen zwischen verschiedenen Sprachen zu unterscheiden, werden die `.dtd` Dateien nicht direkt im

[a] https://addons.mozilla.org/en-US/thunderbird/downloads/file/24181/dom_inspector-2.0.0-fx+tb+sb+sm.xpi
[b] http://developer.mozilla.org/en/docs/Creating_a_Skin_for_Firefox/Getting_Started

Ordner locale abgelegt, sondern in je einem Unterordner pro gewünschter Sprache. Unterordner wären zum Beispiel de für Deutsch und en-US für amerikanisches Englisch.

```
DTD (example.dtd)
   <!ENTITY speichern.label "Speichern">
   <!ENTITY speichern.key "S">

XUL (example.xul)
   <!DOCTYPE windows SYSTEM "chrome://example/locale/example.dtd">
   …
   <menuitem label="&speichern.label;" accesskey="&speichern.key;"/>
```

Abbildung 3.9: Beispiel example.dtd und example.xul Datei

Wie die obige Abbildung einer kurzen .dtd Datei zeigt, ist deren Aufbau denkbar einfach. Für jeden Wert der Stringtabelle wird eine Zeile mit <!ENTITY name "Wert"> eingefügt. Da sich üblicherweise Tastenkürzel am Anfangsbuchstaben der Beschriftung orientieren, müssen auch die Tastenkürzel in der .dtd Datei berücksichtig werden. Die in einer .xul Datei verwendete Stringtabelle muss am Anfang der .xul Datei mittels <DOCTYPE window SYSTEM "Pfad der Datei"> angegeben werden. Danach ist es möglich, durch Voranstellen eines & Zeichens auf den Eintrag in der Stringtabelle zu verweisen.

3.1.6 Preferences

Viele Erweiterungen und auch Thunderbird selbst erlauben es dem Anwender, Einstellungen vorzunehmen. Es ist empfehlenswert, diese Einstellungen von Thunderbird verwalten zu lassen. Thunderbird legt für jeden Anwender eine Datei mit dem Namen prefs.js an und speichert alle benutzerspezifischen Einstellungen in dieser ab. Die prefs.js Datei kann mit einem gewöhnlichen Texteditor gelesen und verändert werden. Zu beachten ist, dass Thunderbird während des Änderns geschlossen sein muss, da dieser immer beim Beenden die aktuellen Werte ablegt und ansonsten jede vorgenommene Änderung überschreiben würde. Pro Einstellung wird in die prefs.js Datei eine Zeile mit Form user_pref("einstellung.name", Wert); eingefügt.

Diese Einstellungen können bei Bedarf von der Benutzeroberfläche oder einer Erweiterung ausgelesen und verändert werden. Die einfachste Weise mit Preferences zu arbeiten ist es, ein prefwindow Element zu benutzen. Dieses ist einem window Element ähnlich, jedoch mit dem Unterschied, dass geänderte Einstellungen automatisch behandelt werden und mit den Preferences abgeglichen werden. Das unten abgebildete Beispiel zeigt das Menü, welches

zum Einstellen von Thunderbird benutzt wird. Durch das Einfügen von `prefpane` Elementen innerhalb eines `prefwindow` Elementes ist es möglich, Karteireiter zu verwenden, welche einfach zu realisieren sind und ungemein zur Steigerung der Übersichtlichkeit betragen.

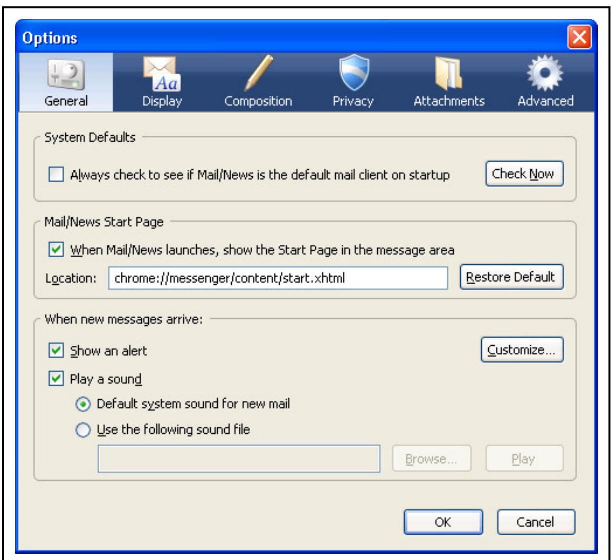

Abbildung 3.10: Preference Window von Thunderbird

Preferences sind eine einfache Möglichkeit kleine, überschaubare Daten abzuspeichern, deren Umfang bekannt ist. Sie sollten jedoch nicht mit einer Datenbank verwechselt werden. Preferences sind zum Beispiel nicht dazu gedacht, alle bekannten Emailadressen abzulegen, sehr wohl allerdings dazu, den Namen des Ordners zu speichern, in welchen Nachrichten verschoben werden sollen, wenn deren Absenderemailadresse unbekannt ist. In den meisten Fällen ist es gewünscht, dass bereits beim Installieren einer Erweiterung Grundeinstellungen vorgenommen worden sind. Aus diesem Grund gibt es den `defaults` Ordner in der Ordnerstruktur eines `.xpi` Paketes. In `defaults` wird ein Unterordner mit dem Namen `preferences` angelegt, in welchem Dateien mit einem beliebigen Namen und `.js` als Endung abgelegt werden können. In diese Dateien können Standardwerte eingetragen werden, welche von Thunderbird herangezogen werden, wenn eine Preference abgefragt wird, welche weder in den Anwender Preferences noch in den Preferences von Thunderbird selbst zu finden ist. Die Struktur einer solchen Datei ist gleich derer von `prefs.js`.

Die Abbildung *3.2: XUL Beispiel Ausgabe* stellt in der `textbox` den Wert der Preference mit dem Namen `example.name` dar. Die dafür verantwortlichen Codezeilen sind:

```
<preferences>
   <preference id="pref_name" name="example.name" type="string"/>
</preferences>

…

<textbox id="name" preftype="char" flex="1" preference="pref name"/>
```
Abbildung 3.11: Beispiel Verwendung von Preferences in example.xul

Zuerst wird durch <preferences> angezeigt, dass eine Aufzählung aller verwendeten Preferences folgt. Für jedes darin angeführte <preference> versucht Thunderbird einen Wert zu finden. Dazu wird der Reihenfolge nach angestrebt, einen Wert in den Preferences des Anwenders, denen von Thunderbird und zuletzt in den defaults zu finden. Wird Thunderbird nicht fündig, wird eine Fehlermeldung ausgegeben. Da das Example anstatt eines pref-window Elementes ein normales window Element verwendet, können zwar Preferences geladen und angezeigt werden, eine Änderung einer Preference wird allerdings nicht in die Einstellungen übernommen.

3.1.7 Install.rdf und chrome.manifest

In jedem .xpi Paket müssen im Hauptverzeichnis eine install.rdf und fallweise eine chrome.manifest Datei abgelegt sein. Diese werden verwendet, um Thunderbird einige für die Installation notwendige Einstellungen mitzuteilen. Einstellungen sind unter anderem der Name des Autors, eine URL, von welcher neue Versionen bezogen werden können, die Version der Erweiterung, für welches Programm die Erweiterung geschrieben wurde und dessen Versionen, und andere mehr. Seit Firefox 1.5 können einige Angaben die ursprünglich in der install.rdf Datei gestanden sind im chrome.manifest abgelegt werden, was auch die empfohlene Vorgehensweise darstellt. Findet Thunderbird beim Installieren keine chrome.manifest Datei vor, wird in der install.rdf Datei nach den benötigten Informationen gesucht. Im Folgenden werden nun die in der install.rdf gespeicherten Informationen näher beschrieben.

<em:id> ist eine eindeutige Identifikationsnummer für eine Erweiterung und muss zwingend angegeben werden. Anhand dieser wird beim Installieren einer Erweiterung entschieden, ob es sich um eine Neuinstallation oder ein Update einer bestehenden Erweiterung handelt. Diese Identifikationsnummer kann auf mehreren Wegen beschafft werden, das einzig Wichtige da-

bei ist, dass diese niemals mehrfach benutzt werden darf. Das Tools GUIDGen[a] von Microsoft generiert passende IDs. Für den seltenen Gebrauch kann unter [b] eine ID generiert werden.

`<em:name>` und `<em:description>` werden verwendet, um die Erweiterung unter `Tools` → `Add-ons` aufzulisten. Der Name muss zwingend vergeben werden, während die Beschreibung wahlweise benutzt werden kann.

`<em:iconURL>` gibt den Pfad für eine Grafik an, welche unter `Tools` → `Add-ons` vor dem Namen einer Erweiterung angezeigt wird. Üblicherweise ist dies eine URL beginnend mit `chrome://`.

`<em:version>` ist verbindlich anzugeben und enthält die Versionsnummer der Erweiterung.

`<em:homepageURL>` beinhaltet die URL zur Homepage der Erweiterung. Auf diese Weise ist es dem Autor einer Erweiterung möglich, auf eine Webseite zu verweisen, welche die aktuelle Version der Erweiterung und zusätzliche Informationen beinhaltet.

`<em:updateURL>` wird verwendet, um einen Verweis auf eine Updatequelle zu setzen. Thunderbird überprüft beim Starten, ob eine neue Version der Erweiterung vorhanden ist und empfiehlt, wenn nötig, ein Update.

`<em:updateKey>` ist seit Gecko 1.9 zwingend erforderlich, um sicherzustellen, dass Erweiterungen digital signiert sind. Dadurch ist es einem Autor möglich, offizielle Updates zu signieren, was dazu führt, dass ausgeschlossen werden kann, dass jemand gefälschte Updates verwendet, um schadhafte Programme einzuschleusen. Mit dem Programm McCoy[c] ist es möglich, den `updateKey` zu erstellen und Updates zu signieren.

`<em:targetApplication>` muss zwingend angegeben werden und beschreibt, für welche Anwendung die Erweiterung gedacht ist. Dazu hat nicht nur jede Erweiterung, sondern auch jede Anwendung eine eindeutige ID. Das unten abgebildete Beispiel zeigt die ID von Thunderbird.

---

[a] http://www.microsoft.com/downloads/details.aspx?familyid=94551f58-484f-4a8c-bb39-adb270833afc&displaylang=en
[b] http://www.guidgen.com/
[c] http://developer.mozilla.org/en/docs/McCoy

`<em:minVersion>` und `<em:maxVersion>` beschreiben die minimale bzw. maximale Versionsnummer der Anwendung. Da bei jedem Update Änderungen an der API von Thunderbird vorgenommen werden, ist es wichtig, genauestens anzugeben, für welche Version die Erweiterung funktioniert.

```
<?xml version="1.0"?>
<RDF:RDF  xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <RDF:Description RDF:about="urn:mozilla:install-manifest">
    <em:id>{F62DA64F-3337-47bc-9951-9931C01DDADE}</em:id>
    <em:name>Example</em:name>
    <em:description>This is the example!</em:description>
    <em:version>1.0</em:version>
    <em:creator>Christian Haider</em:creator>
    <em:homepageURL>http://localhost</em:homepageURL>
    <em:iconURL>chrome://example/skin/icons.png</em:iconURL>
    <em:updateURL>http://localhost/update.rdf</em:updateURL>
    <em:targetApplication>
      <RDF:Description>
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>3.*</em:maxVersion>
      </RDF:Description>
    </em:targetApplication>
  </RDF:Description>
</RDF:RDF>
```

Abbildung 3.12: Beispiel install.rdf

Mit dem oben angeführten `install.rdf` sind alle Informationen, die eine Erweiterung eindeutig beschreiben, gegeben. Für eine erfolgreiche Installation muss Thunderbird jedoch auch mitgeteilt werden, wie die zu installierenden Dateien im `.xpi` Paket verteilt sind, und welche Dateien im CHROME registriert werden müssen. Dies wird in der `chrome.manifest` Datei beschrieben. Das unten angeführte Beispiel ist eine kurze `chrome.manifest` Datei, welche alle in den Unterordnern abgelegten Dateien aufzählt.

```
content example jar:chrome/example.jar!/content/
overlay chrome://messenger/content/messenger.xul
chrome://example/content/overlay.xul
skin example classic/1.0 jar:chrome/example.jar!/skin/classic/
locale example en-US jar:chrome/example.jar!/locale/en-US/
locale example de jar:chrome/example.jar!/locale/de/
```

Abbildung 3.13: Beispiel chrome.manifest

Mit `content` in der ersten Zeile wird angegeben, dass der Paketname und der Pfad zur Benutzeroberfläche einer Erweiterung folgen. „`example`“ ist der Paketname der Erweiterung

und wird beim Registrieren von Dateien im CHROME immer hinter `chrome://` angefügt. In diesem Beispiel setzt sich der Pfad aus zwei Teilen zusammen. Zuerst wird mit `jar:` angegeben, dass sich innerhalb des `.xpi` Paketes eine komprimierte Datei befindet, welche die Benutzeroberfläche beinhaltet. Der zweite Teil, `chrome/example.jar!/content/`, ist der relative Pfad zur Benutzeroberfläche. Das ! markiert dabei die komprimierte Datei.

Die zweite Zeile weist Thunderbird durch `overlay` an, ein Overlay zu registrieren. Der erste Pfad gibt die URL an, auf welche das mit dem zweiten Pfad angegebenen Overlay platziert werden soll.

Mittels `skin` wird angegeben, welches Skin mit der Erweiterung installiert wird. Dazu wird abermals der Paketname gefolgt von dem Skinnamen angegeben. Durch Verwenden des Standardskins von Thunderbird mit dem Namen `classic/1.0` kann erreicht werden, dass die Erweiterung automatisch beim Ändern des Standardskins mit angepasst wird. Die letzte Angabe dieser Zeile ist der relative Pfad zum `skin` Ordner. In diesem Fall ist dieser in der komprimierten Datei abgelegt.

Durch `locale` wird angegeben, dass eine Erweiterung in einer bestimmten Sprache verfügbar ist. Nach `locale` folgt der Paketname und im Anschluss die gewünschte Sprache. Die Beispielerweiterung ist in zwei Sprachen verfügbar, durch `en-US` in amerikanischem Englisch und durch `de` in Deutsch. Wie bereits bei den vorhergehenden Optionen ist zuletzt wieder der relative Pfad anzugeben.

3.1.8 Packaging

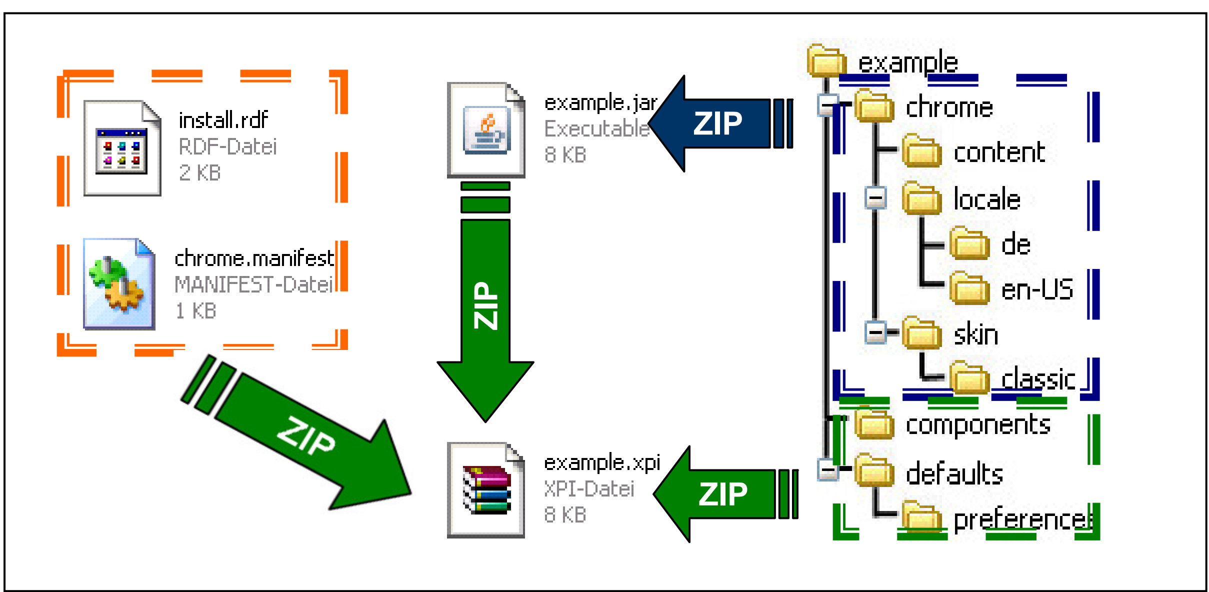

Abbildung 3.14: .xpi Paket

Alle für eine Erweiterung benötigten Dateien werden in einer `.xpi` Datei zusammengefasst und können damit als Paket verteilt werden. Eine solche `.xpi` Datei ist eine komprimierte Datei, welche die Ordnerstruktur einer Erweiterung beinhaltet. Die oben abgebildete Grafik zeigt die Ordner und Dateien der Beispielerweiterung, und wie diese in das `.xpi` Paket eingefügt werden müssen.

Zuerst werden alle Unterordner und die darin enthaltenen Dateien des `chrome` Ordners komprimiert und in eine Datei mit der Endung `.jar` gespeichert. Zu beachten ist, dass es sich dabei nicht, wie die Dateiendung vermuten lässt, um ein Java Archive handelt und darin Java Klassen abgelegt sind. Empfehlenswert ist es, für die `.jar` Datei als Dateinamen den Namen der Erweiterung zu wählen und nochmals zu prüfen, ob dieser in den Dateien `install.rdf` und `chrome.manifest` richtig eingesetzt wurde. Die erstellte `.jar` Datei wird im Anschluss mit den Ordnern `components` und `defaults` und den Dateien `install.rdf` und `chrome.manifest` abermals komprimiert und in eine Datei mit einem beliebigen Namen und der Endung `.xpi` abgespeichert. Diese `.xpi` Datei ist nun ein fertiges Paket und kann durch die in Kapitel *2.2 Thunderbird* beschriebenen Varianten installiert werden. Um das Erstellen eines `.xpi` Paketes zu beschleunigen, bietet sich die Verwendung einer Batch Datei an, wie sie in Kapitel *8.1.2 Batch Datei für .xpi Paket* angeführt ist.

3.1.9 XPCOM

Mit dem Einsatz von XPCOM (Cross Platform Component Object Module) wird versucht, die Wiederverwendbarkeit und Flexibilität von Klassen, die Thunderbird benutzt oder eine Erweiterung implementiert, zu erhöhen. Dies geschieht, indem ein Interface erstellt wird, das alle öffentlichen Funktionen einer Klasse beinhaltet, und die Klasse ausschließlich über dieses Interface benutzt wird. Die dabei entstehenden Klassen werden Components genannt. Das Konzept ist ähnlich dem von Microsoft entwickelten COM, jedoch mit dem Vorteil, dass XPCOM auf allen Betriebssystemen, für welche Mozilla verfügbar ist, verwendet werden kann. Eine Component wird erst bei Bedarf, also zur Laufzeit einer Anwendung, geladen und kann, wenn nötig, auch wieder aus dem Speicher entfernt werden. Kombiniert mit der strikten Trennung der eigentlichen Implementierung des Interfaces wird es ermöglicht, eine Klasse im Nachhinein zu ändern oder auszutauschen, ohne die tatsächliche Anwendung neu zu kompilieren. Dies hat den großen Vorteil, dass Components von Anwendungen benutzt werden können, ohne dass diese der Anwendung bereits bei der Implementierung bekannt waren. Zum Beispiel könnten damit die bekannten Dateitypen eines Texteditors durch nachträgliches Hinzufügen einer neuen Component erweitert werden, ohne den gesamten Texteditor neu kompilieren zu müssen. Das in dieser Diplomarbeit besprochene Framework nutzt diese Eigenschaft, um zusätzliche Pluggies einhängen zu können, ohne dass diese dem Framework zuvor bekannt waren.

Für Erweiterungen, die in JS geschrieben sind, ist es nicht zwingend erforderlich, die implementierten Klassen XPCOM kompatibel zu gestalten. Um die in Thunderbird bereitgestellte API verwenden zu können, ist es jedoch auch für JS Entwickler notwendig, die hinter XPCOM stehende Technologie zu verstehen [TuOe03].

3.1.9.1 Interface

Mit einem Interface wird die öffentliche Schnittstelle einer Component beschrieben. Diese Herangehensweise ist in der objektorientierten Programmierung nichts Neues, jedoch wird bei XPCOM dieser Ansatz noch verfeinert. Wenn ein herkömmliches Programm eine Funktion einer Bibliothek verwenden möchte, muss die Bibliothek bereits beim Kompilieren verfügbar sein. Bei XPCOM wird durch XPIDL (Cross Platform Interface Definition Language) ermöglicht, diese Bindung erst zur Laufzeit vorzunehmen. Dadurch kann zum Beispiel ein Zeichen-

programm, mit dem ausschließlich Quadrate gezeichnet werden können, zusätzliche Schablonen laden, um auch Kreise zeichnen zu können.

Auf diese Weise kann ein Programmierer seine Components als Blackbox entwerfen, und die eigentliche Implementierung nach außen geheim halten. Einer Anwendung, die diese Components benutzt, sind nur die nach außen definierten Funktionen bekannt. Wird innerhalb einer Component ein Fehler korrigiert oder eine Performanceoptimierung vorgenommen, ist dies für alle Anwendungen, die die Component benutzen, irrelevant, solange sich das verwendete Interface nicht verändert.

3.1.9.2 Components

Es würde für einen einzigen Programmierer eine unmögliche Aufgabe darstellen, die rund 4 Millionen Codezeilen von Mozilla zu überblicken. Durch das Aufteilen des Quellcodes in kleine Pakete entsteht eine Granularisierung, die es ermöglicht, die Verantwortung für verschiedene Bereiche an verschiedene Gruppen von Programmierern zu übergeben. Die für das Netzwerk verantwortlichen Teile von Thunderbird werden zum Beispiel in einer Bibliothek namens Necko zusammengefasst und als Components implementiert.

Unter einer Component versteht man eine mit XPCOM kompatible Klasse. Diese kann sowohl in JS, C++ und inzwischen auch in Java und Python implementiert sein. Um eine Klasse XPCOM kompatibel zu gestalten, muss diese vom Basisinterface `nsISupports` abgeleitet werden. Dieses Interface stellt die Funktionen `AddRef`, `Release` und `QueryInterface` zur Verfügung.

Die 3 Funktionen werden von XPCOM zur Verwaltung der Objekte benötigt. Durch `AddRef` und `Release` kann die Lebenszeit einer Component gesteuert werden. Dazu beinhaltet die Component einen Referenzzähler. Mit jeder Referenz, die auf ein Objekt angelegt wird, erhöht sich der Referenzzähler um eins und verringert sich, wenn die Referenz gelöscht wird. Das bedeutet, dass der Referenzzähler 0 erreicht, wenn keine Referenz mehr auf das Objekt besteht, und dieses nicht mehr benutzt werden kann. In dem Fall beendet das Objekt seine Lebenszeit und gibt seinen Speicher wieder frei. Mit `QueryInterface` stellt XPCOM zur Laufzeit fest, ob eine Component ein gewünschtes Interface implementiert und damit die Funktionen des Interfaces bereitstellt.

Components, die in C++ geschrieben wurden, müssen vor deren Benutzung kompiliert werden und sind somit an ein Betriebssystem gebunden. Diese werden in binären Bibliotheken gespeichert, welche üblicherweise in Windows mit `.dll` und Linux mit `.dso` enden. Werden in einer solchen Bibliothek mehrere zusammengehörige Components gespeichert, spricht man von einem Modul. XPCOM bietet neben dem Vorteil der Wiederverwendbarkeit noch die Möglichkeit, Klassen aus einer anderen Programmiersprache zu benutzen. So wurden die meisten in Thunderbird zur Verfügung gestellten Components in C++ entwickelt und können von JS Erweiterungen genutzt werden. Dies wird durch die in Thunderbird integrierte Technologie mit dem Namen XPConnect ermöglicht. XPConnect ist für einen Entwickler in den meisten Fällen völlig transparent und übernimmt die Umwandlung der Funktionsaufrufe von JS auf C++ und vice versa.

Durch Vererbung ist es möglich, dass eine Component die Funktionalität einer andern Component übernimmt, ohne dass diese die gesamte Funktionalität erneut implementieren muss. Wie bereits erwähnt, werden alle Components von `nsISupports` abgeleitet und verfügen über dessen Funktionen. Eine Component, die nur `nsISupports` implementiert, wäre denkbar nutzlos, weshalb zusätzliche Interfaces hinzugefügt werden, und somit die Component auch Funktionen zusätzlicher Interfaces beherrscht. Um diese zusätzlichen Funktionen verwenden zu können, muss es zur Laufzeit möglich sein, ein Objekt einer Component zu fragen, ob es ein bestimmtes Interface unterstützt. Dies wird durch die `QueryInterface` Methode ermöglicht.

Um jede Component und jedes Interface eindeutig identifizieren zu können, werden diese mit einer 128 bit langen ID, üblicherweise `CID` genannt, versehen. Die `CID` wird zwar in hexadezimaler Form geschrieben, es wäre allerdings dennoch umständlich, diese beim Programmieren zu verwenden. Aus diesem Grund wird jeder Component zusätzlich zur `CID` eine für den Programmierer leichter verwendbare Zeichenkette, mit dem Namen `ContractID,` zugewiesen. Diese Zeichenkette kann verwendet werden, um ein Objekt einer Component von XPCOM anzufordern [TuOe03]. Im Appendix werden unter *8.1.3 JS Beispiel Component* und *8.1.4 C++ Beispiel Component* zwei kurze Beispiele derselben Component, sowohl in JS als auch in C++ geschrieben, gegeben. Die Component stellt eine Funktion mit dem Namen `allCapital` zur Verfügung.

3.1.9.3 XPIDL

So weit wurde geklärt, dass XPCOM auf eine strikte Trennung von Interfaces und Programmcode setzt, und dass jede Component von `nsISupports` erbt. Bisher unbekannt ist jedoch, wie es durch `QueryInterface` möglich ist, zur Laufzeit zu erfragen, welche Interfaces eine Component implementiert und welche Methoden, Attribute und Konstanten damit unterstützt werden. Dazu bedient sich XPCOM der Beschreibungssprache XPILD (Cross Platform Interface Definition Language). XPIDL ist eine Abwandlung von CORBA IDL (Interface Definition Language) und unterstützt lediglich eine Teilmenge der von CORBA ILD gebotenen Möglichkeiten. Da XPCOM ausschließlich innerhalb einer Anwendung mit anderen Components kommunizieren muss, ist der Teil von CORBA ILD, der für RPC (Remote Procedure Call) zuständig ist, nicht notwendig [Emm97].

Leider birgt die Verwendung von XPIDL auch zwei entscheidende Nachteile. Zum einen ist keine Mehrfachvererbung von Klassen möglich, eine Mehrfachvererbung von Interfaces jedoch schon (ähnlich JAVA). Zum anderen unterstützt XPIDL das Überlagern von Methoden nicht. Das Resultat ist, dass nicht der Methodenkopf, sprich der Methodenname inklusive allen übergebenen Attribute und deren Typen, eindeutig sein muss, sondern bereits der Methodenname alleine. Trotz dieser beiden Einschränkungen ist der Einsatz von XPIDL gerechtfertigt, da er für XPCOM eine einfache Lösung darstellt, für alle Programmiersprachen Beschreibungen von Components zur Verfügung zu stellen.

Ein Interface, das von einer Component implementiert werden kann, wird in einer `.idl` Datei beschrieben. Die enthaltenen Informationen werden von einer kleinen Anwendung namens xpidl in eine binäre Typ-Bibliothek mit der Endung `.xpt` umgewandelt, in welcher die notwendigen Einstiegspunkte in Funktionen und Attribute für Programmiersprachen, wie JS, abgelegt sind. Diese Einstiegspunkte werden von XPCOM dazu benutzt, den Inhalt einer Component anzusprechen. Lediglich für Components die in C++ geschrieben werden, werden diese Typ-Bibliotheken nicht benötigt. Die folgende Abbildung zeigt, wie der Inhalt einer `.idl` Datei aussehen könnte.

```
#include "nsISupports.idl"
[scriptable, uuid(6A971889-15DA-4bbf-87EF-B5603DD75DAC)]
interface nsIExample : nsISupports{
  string allCapital(in string s);
};
```

Abbildung 3.15: Beispiel .idl Datei

Durch `#include "nsISupports.idl"` wird xpidl mitgeteilt, den Inhalt der `nsISupports.idl` Datei zu importieren. Dies ist notwendig, da das Interface `nsIExample` von `nsISupports` abgeleitet wird und somit alle Funktionen von diesem erbt. Durch das Anführen von `scriptable` in der zweiten Zeile wird XPCOM dazu angewiesen, das Interface über XPConnect für JS zugänglich zu machen. Dies ist für alle Interfaces möglich, die ausschließlich die von XPIDL unterstützten Datentypen[a] verwenden. Ist es notwendig, einen sogenannten „`native`" Type zu verwenden, ein solcher wäre zum Beispiel der in C++ vorhandene `vector`, kann entweder durch Löschen von `scriptable` das gesamte Interface oder durch Voranstellen von `[noscript]` in der betroffenen Zeile nur die betroffene Funktion als nonscriptable deklariert werden. `string allCapital(in string s)` ist die Deklaration einer Funktion mit `string` als Rückgabetyp. Für jedes Attribut muss der Typ, seine Richtung und ein Name angegeben werden. Bei der Richtung wird zwischen Eingangs- `(in)`, Ausgangs- `(out)` oder Ein/Ausgangsparameter `(inout)` unterschieden.

3.1.9.4 Factory

Da es mit XPIDL möglich ist, Components, die bis zur Laufzeit unbekannt waren, zu nutzen und in ein Programm einzubinden, liegt es nahe, dass XPCOM auch die Erzeugung dementsprechender Objekte reguliert. Damit ist sichergestellt, dass der Benutzer einer Component immer darauf vertrauen kann, dass dieses richtig initialisiert wurde und zur Benutzung bereit steht. Um dies zu bewerkstelligen, benutzt XPCOM das unter dem Namen Factory bekannte Design Pattern. Beim Factory Design Pattern wird niemals direkt ein neues Objekt der gewünschten Klasse erzeugt, sondern ein Objekt von einer „Fabrik" angefordert. Diese „Fabrik" erzeugt das gewünschte Objekt und kapselt sowohl die Erzeugung als auch die Initialisierung des Objektes. Dies hat den Vorteil, dass der Benutzer, der das Objekt anfordert, nichts über das tatsächliche Objekt wissen muss, sondern nur über ein Interface auf dieses zuzugreifen braucht.

[a] http://www.mozilla.org/scriptable/xpidl/idl-authors-guide/rules.html#types

Die bei XPCOM benutzte „Fabrik“ muss sowohl das Interface `nsIFactory` als auch `nsI-Supports` implementieren und ist damit selbst eine Component. Es gibt zwei Arten von Components und diesen zugehörige „Fabriken“: Zum einen die „normalen“ Components, von welchen beliebig viele Objekte erzeugt werden können, und zum anderen die so genannten Services. Von zuletzt genannten Components wird ein einziges Objekt erzeugt und bei erneutem Anfordern wieder auf dieses verwiesen. Dieser Mechanismus ist auch als Singleton Design Pattern bekannt.

3.1.9.5 Module

Jede Component, die in XPCOM verwendet werden kann, muss zuvor in XPCOM registriert worden sein. Dazu muss zusätzlich zur „Fabrik“ und der Component selbst noch ein dritter Teil implementiert werden, welcher vom Interface `nsIModul` erbt. Dieser Teil ist ebenfalls eine Component und implementiert die Methode `NSGetModul`, welche XPCOM verwendet, um ein Objekt der „Fabrik“ zu erzeugen. Diese „Fabrik“ kann im Anschluss von XPCOM genutzt werden, wenn ein Objekt der Component angefordert wird.

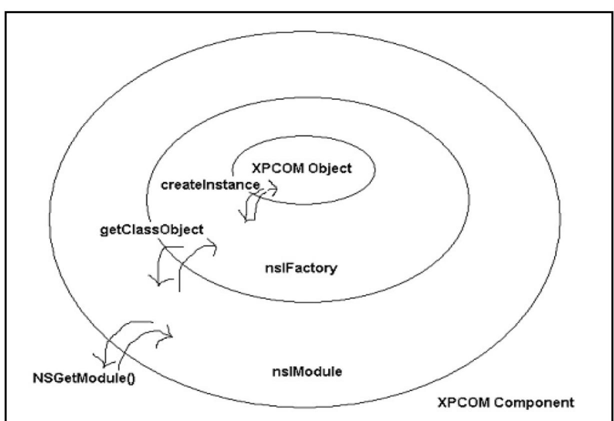

Abbildung 3.16: Peel View of XPCOM Component Creation [TuOe03]

So wie in der Grafik zu sehen ist, ist der der Aufbau einer XPCOM Component in Schichten, ähnlich einer Zwiebel, vorstellbar. Durch Zuhilfenahme der jeweils weiter außen liegenden Schicht kann auf die Methoden der jeweils weiter innen liegenden Schicht zugegriffen werden. Wie aus der Abbildung schön hervorgeht, stellt `NSGetModule` den einzigen Einstiegspunkt für eine Component dar.

3.2 Environment für einfache Projekte

Dieses Kapitel beschreibt, wie eine Entwicklungsumgebung unter Windows XP für einfache, in JS implementierte, Erweiterungen gestaltet werden kann. Da es für das Schreiben von Erweiterungen keine Out-Of-The-Box Entwicklungsumgebung gibt, werden einige hilfreiche, frei verfügbare Werkzeuge genannt, welche das Entwickeln komfortabler gestalten.

Zu Beginn sollte die aktuelle Version von Thunderbird[a] heruntergeladen und installiert werden. Das Programmieren von Erweiterungen kann in einem einfachen Texteditor erfolgen, deshalb bietet sich ein frei verfügbares Programm, wie Notepad++[b] mit Syntax Highlighting, an. Es ist empfehlenswert, bereits zu Beginn die gesamte Ordnerhierarchie für die zu schreibende Erweiterung anzulegen. Diese kann aus dem, in Kapitel *3.1 Einführung Mozilla Programmierung* vorgestellten Beispiel, übernommen werden. Um aus der fertig programmierten Erweiterung ein `.xpi` Paket zu erstellen, kann die in Windows integrierte Zip-Funktion benutzt werden. Ein besseres Ergebnis kann jedoch mit dem frei verfügbaren 7-Zip Programm[c] erzielt werden. Dieses kann über die Kommandozeile gesteuert werden und erlaubt somit das Arbeiten mit Batch Dateien. Siehe dazu Kapitel *8.1.2 Batch Datei für .xpi Paket*. Damit 7-Zip von der Kommandozeile aus genutzt werden kann, muss entweder die Pfad-Variable von Windows angepasst, oder die Datei 7-zip.exe in den im Windows Ordner befindlichen `System32` Ordner kopiert werden.

Erweiterungen, die lediglich auf bestehende Interfaces aufbauen und keine Components mit neuen Interfaces erstellen, können bereits mit den wenigen, oben genannten Programmen geschrieben werden. Sollten eigene Interfaces benötigt werden, müssen die für Thunderbird benötigten binären Typ-Bibliotheken aus den in `.idl` Dateien vorliegenden Schnittstellenbeschreibungen erstellt werden. Dazu sind noch einige zusätzliche Schritte durchzuführen.

Hilfreich ist es, einen Ordner mit dem Namen `idl` zu erstellen, und das verfügbare Development Kit[d] in dieses zu entpacken. Dieses beinhaltet die wichtigsten in Thunderbird verwendeten, `.idl` Dateien und das zum Umwandeln benötigte Programm `xpidl`. Leider wurde `xpidl` nicht statisch kompiliert, weshalb zwei zusätzliche Bibliotheken installiert werden müssen. Diese beiden benötigten Dateien heißen „`libIDL-0.6.dll`“ und „`glib-1.2.dll`”

[a] http://www.mozilla.com/thunderbird/
[b] http://notepad-plus.sourceforge.net/
[c] http://www.7-zip.org/
[d] http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases/1.8.0.4/sdk/gecko-sdk-win32-msvc-1.8.0.4.zip

und sind gemeinsam in einem Paket[a] enthalten. Die Dateien müssen in denselben Ordner wie `xpidl` kopiert werden. Danach ist es möglich, aus `.idl` Dateien `.xpt` Dateien zu erzeugen. Wichtig ist, dass beim Starten von `xpidl` mit der Option `-I"Pfad"` der Pfad zu den im Development Kit enthaltenen `.idl` Dateien angegeben wird. Wird dieser Pfad nicht angegeben, ist es `xpidl` nicht möglich, neue Interfaces von den bereits bestehenden abzuleiten. Da jedes Interface direkt oder indirekt über weitere Interfaces von `nsISupports` abgeleitet werden muss, würde dies unweigerlich zu einem Fehler führen. Ist eine `.idl` Datei syntaktisch nicht korrekt, führt dies in den meisten Fällen zur Ausgabe einer sprechenden Fehlermeldung. Leider sind dem Autor auch Fälle bekannt, bei denen `xpidl` von Windows unter Angabe einer Speicherverletzung geschlossen wird. Es ist daher empfehlenswert, `.idl` Dateien sorgfältig zu schreiben.

3.3 Environment für komplexe Projekte

Häufig ist eine, wie in Kapitel *3.2 Environment für einfache Projekte* vorgestellte Entwicklungsumgebung, nicht ausreichend. Einer der häufigsten Gründe dafür ist, dass nicht ausschließlich JS verwendet wird, sondern auch C++ zum Einsatz kommt. Dies hat zwei Konsequenzen. Zum einen müssen solche Erweiterungen kompiliert werden, bevor sie in ein `.xpt` Paket verpackt werden können. Zum anderen ist es in C++ notwendig, die Header Dateien aller benutzten Interfaces zu importieren. Diese sind zwar zum Großteil in dem verfügbaren XUL-Runner[b] enthalten, werden jedoch „non-frozen“ Interfaces benutzt, müssen diese von Hand in die Entwicklungsumgebung eingebunden werden. Unter „frozen“ Interfaces werden all jene Schnittstellen verstanden, die in Mozillas API als fixiert gelten. Bei diesen Interfaces kann ein Entwickler davon ausgehen, dass diese nicht verändert werden. Grundsätzlich sollten ausschließlich „frozen“ Interfaces benutzt werden. Nichtsdestotrotz sind viele nützliche Components lediglich über „non-frozen“ Interfaces verfügbar. Eine Änderung führt unter diesen Umständen dazu, dass die geschriebene Erweiterung an das neue Interface angepasst werden muss.

Da Erweiterungen, die in C++ geschrieben sind, pro gewünschten Betriebssystem kompiliert werden müssen, bietet es sich an, diese so zu gestalten, dass deren Kompilierung gemeinsam mit jener von Thunderbird erfolgen kann. Die folgenden Unterkapitel erläutern, wie dieser sogenannte Build-Prozess funktioniert, und welche Anforderungen dafür an die Entwick-

[a] http://ftp.mozilla.org/pub/mozilla.org/mozilla/source/wintools.zip
[b] http://ftp.mozilla.org/pub/mozilla.org/xulrunner/nightly/latest-trunk/

lungsumgebung gestellt werden. Das Ergebnis ist eine Entwicklungsumgebung für Windows, mit welcher sowohl Thunderbird als auch zusätzliche Erweiterungen kompiliert werden können [Mec04].

Die in Kapitel *3.1 Einführung Mozilla Programmierung* vorgestellte Erweiterung wird so angepasst, dass sich diese problemlos sowohl unter Windows als auch unter Linux und MAC OSX kompilieren lässt.

3.3.1 Installationen

Im Gegensatz zu Linux beinhaltet Windows nicht von vornherein einen C++ Kompiler. Daher ist es notwendig, einige Programme zu installieren, bevor Thunderbird oder eine Erweiterung kompiliert werden kann. Microsoft bietet einen frei verfügbaren C++ Kompiler[a] an, welcher für den Build-Prozess geeignet ist. Zusätzlich muss das verfügbare Software Development Kit[b] installiert werden. Der Build-Prozess verwendet das GNU Make Programm, um auf allen Betriebsystemen kompilierbar zu sein. Leider ist dieses Programm neben einer Vielzahl anderer Werkzeuge, die für den Build-Prozess benötigt werden, nicht in Windows enthalten. Abhilfe schafft ein Softwarepaket mit dem Namen `mozilla-build`[c], welches alle zusätzlich benötigten Programme beinhaltet und gemeinsam installiert. Um die Programme benutzen zu können, muss passend zum installierten C++ Kompiler eine der Verknüpfungen, die im Installationsordner von `mozilla-build` liegen, gestartet werden.

3.3.2 CVS

Der Quellcode von Thunderbird steht, wie jener von zahlreichen anderen aus dem Mozilla Projekten entstandenen Anwendungen, zum Download zur Verfügung. Es gibt zwei Möglichkeiten, diesen zu erhalten. Zum einen kann der Quellcode von offiziellen Releases als komprimierte Datei[d] heruntergeladen werden. Da Thunderbird kontinuierlich weiterentwickelt wird, kann es notwendig sein, den aktuellsten Quellcode zu verwenden, in welchem mitunter bereits zahlreiche Fehler behoben wurden. Da nicht jede Änderung zu einem neuen Release führt, muss CVS (Concurrent Versioning System) verwendet werden, was die zweite Möglichkeit darstellt. CVS erlaubt es vielen Entwicklern, gemeinsam an einem Programm zu ar-

[a] http://msdn.microsoft.com/vstudio/express/visualc/download/
[b] http://www.microsoft.com/downloads/details.aspx?familyid=0baf2b35-c656-4969-ace8-e4c0c0716adb
[c] http://ftp.mozilla.org/pub/mozilla.org/mozilla/libraries/win32/MozillaBuildSetup-1.2.exe
[d] ftp://ftp.mozilla.org/pub/mozilla.org/thunderbird/releases/

beiten. Dazu wird auf einem gemeinsamen Server immer der aktuellste Quellcode in einem sogenannten Repository bereitgestellt. Jeder Entwickler kann dieses auf seinen lokalen Computer herunterkopieren und im Anschluss mit dieser lokalen Kopie arbeiten. Nach erfolgreichem Ändern einer Programmstelle wird das Original, welches am Server liegt, durch das Einspielen der lokalen Kopie abgeändert. Diese Vorgehensweise erfordert viel Disziplin bei den Entwicklern, da Fehler in einem Programmstück Auswirkungen auf das gesamte Programm haben könnten. Damit keine unbefugten Änderungen am offiziellen Quellcode vorgenommen werden können, benötigt man zum Ändern spezielle Berechtigungen. Für das Abfragen der aktuellsten Version wird kein Benutzername gefordert. Für Entwickler, die aktiv am Thunderbird Projekt mitwirken, wird ein Benutzer angelegt, welcher die Berechtigung zum Ändern erhält. Wird ein Fehler in Thunderbird behoben, wird diese Änderung als Patch bezeichnet. Bevor nun ein solcher Patch in den offiziellen Quellcode eingefügt wird, wird das Durchlaufen eines Review-Prozesses verlangt, der vorsieht, dass mindestens ein zweiter Entwickler den Patch begutachtet.

Um CVS zu verwenden, müssen folgende Schritte durchgeführt werden:

1. Starten der Verknüpfung im `mozilla-build` Ordner.
2. Mittels „`cd "Pfad"`" in den Ordner wechseln, in dem der Quellcode abgelegt werden soll.
3. Eingabe von „`cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot co mozilla/client.mk`". Durch diesen Aufruf wird die Make Datei `client.mk` vom Repository heruntergeladen.
4. Eingabe von „`cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot co mozilla/mail/config/mozconfig`". Dadurch wird die im Repository befindliche Datei `mozconfig` heruntergeladen.
5. Erst durch die in Schritt 3 und 4 heruntergeladenen Dateien ist es CVS möglich, alle für Thunderbird benötigten Dateien zu bestimmen. Mit dem Aufruf von „`cd mozilla`", gefolgt von „`make -f client.mk checkout MOZ_CO_PROJECT=mail`", wird der gesamte Quellcode, inklusive aller für das Kompilieren benötigten Dateien, heruntergeladen.

Um die herunterzuladende Datenmenge zu begrenzen, ermöglicht es CVS, nur die geänderten Dateien herunterzuladen und die lokal gespeicherten Dateien zu aktualisieren. Um dies zu bewerkstelligen müssen, die Schritte 1,2 und 5 erneut ausgeführt werden.

3.3.3 Erstes Build

Es ist empfehlenswert, Thunderbird beim ersten Versuch ohne zusätzliche, eigene Erweiterungen zu kompilieren. Damit kann ausgeschlossen werden, dass vergeblich ein Fehler in der eigenen Erweiterung gesucht wird, obwohl eigentlich die Entwicklungsumgebung falsch eingerichtet ist.

Der Build-Prozess ist sehr flexibel und erlaubt viele Einstellungen [Mec04]. Dies hat leider auch den Nachteil, dass einige Einstellungen zwingend vorzunehmen sind. Bevor Thunderbird kompiliert werden kann, muss eine `.mozconfig` Datei im Hauptverzeichnis des Quellcodes angelegt werden, aus welcher wichtige Parameter für den Build-Prozess ausgelesen werden. Der Windows Explorer erlaubt es nicht, Dateinamen mit „.“ beginnend zu benennen, weshalb das Anlegen der `.mozconfig` Datei entweder durch „Speichern unter“ aus einem Texteditor, oder mittels der Eingabe von „`touch .mozconfig`“ in die Kommandozeile von `mozilla-build` erfolgen muss. Die folgende Abbildung zeigt ein Beispiel einer `.mozconfig` Datei.

```
# My first mozilla config
. $topsrcdir/mail/config/mozconfig
mk_add_options MOZ_OBJDIR=/d/mozilla/mozilla_build
ac_add_options --enable-application=mail
ac_add_options --enable-extensions=default
ac_add_options --enable-optimize
ac_add_options --disable-debug
```

Abbildung 3.17: .mozconfig Datei

Die erste Zeile beginnt mit einer „#“ und ist somit ein Kommentar. Erst die zweite Zeile beinhaltet eine für den Build-Prozess relevante Anweisung. Durch diese Zeile wird veranlasst, dass alle Einstellungen der Standard `mozconfig` Datei von Thunderbird gelesen und gesetzt werden. Diese Grundeinstellungen können im Anschluss in der vom Entwickler angelegten `.mozconfig` Datei überschrieben werden. Mit dem Build-Prozess ist es möglich, mehrere Versionen von Thunderbird, ausgehend vom selben Quellcode, zu erstellen. Dazu ist es notwendig, die kompilierten Dateien getrennt vom Quellcode abzuspeichern. Dazu muss ein Ver-

zeichnis mit der Option „mk_add_options MOZ_OBJDIR=/Pfad“ in der .mozconfig Datei angegeben werden, in welches die kompilierten Dateien abgelegt werden sollen. Das Verzeichnis wird Objekt-Verzeichnis genannt. Die Option „mk_add_options ac_add_options --enable-application=mail“ weist den Build-Prozess dazu an, Thunderbird zu kompilieren. Wird in der Option „mail“ durch „browser“ ersetzt, und ist gleichzeitig der passende Quellcode verfügbar, würde der Build-Prozess versuchen, Firefox zu kompilieren. Der Build-Prozess ermöglicht es, in einem Vorgang sowohl Thunderbird selbst als auch zusätzliche Erweiterungen, wie den Terminkalender Lightning, zu kompilieren. Mit der Option „ac_add_options --enable-extensions=default“ wird eingestellt, dass nur die Standarderweiterungen kompiliert werden sollen. In der aktuellen Version ist das lediglich die Erweiterung Talkback, die Fehlerberichte an einen zentralen Server weiterleitet. Die Optionen „ac_add_options --enable-optimize“ und „ac_add_options --disable-debug“ geben an, dass der Kompiler versuchen soll, den Binärcode zu optimieren und keine Debug Informationen einzufügen. Dies erhöht die Geschwindigkeit des erzeugten Thunderbirds. Um eine für Debug geeignete Version von Thunderbird zu erstellen, sollten diese Optionen durch „ac_add_options --disable-optimize“ und „ac_add_options --enable-debug“ ersetzt werden. Dadurch werden Quellcodeinformationen in das Kompilat eingefügt, was erlaubt, mit einem Debugger Fehlerdiagnosen durchzuführen.

Wenn die .mozconfig Datei die richtige Konfiguration enthält, und sich die Kommandozeile im Hauptverzeichnis des Quellcodes befindet, kann durch Aufrufen von „export MOZCONFIG=./.mozconfig“, gefolgt von „make -f client.mk build“, der Build-Prozess gestartet werden. Dieser überprüft zu Beginn, sowohl ob alle benötigten Programme installiert sind, als auch einen Großteil, der in der .mozconfig Datei vorgenommenen Einstellungen. Das eigentliche Kompilieren kann in Abhängigkeit von den am Computer verfügbaren Ressourcen einige Stunden dauern. Sollte der Quellcode seit dem letzten Kompilieren aktualisiert worden sein, überprüft Make, ob eine Datei seit dem letzen Build-Prozess geändert wurde und kompiliert nur geänderte Dateien.

3.3.4 Build-Prozess mit Erweiterung

In diesem Kapitel wird erläutert, wie die in Kapitel *3.1 Einführung Mozilla Programmierung* verwendete Beispielerweiterung an den Build-Prozess von Thunderbird angepasst werden kann.

```
# My first mozilla config
. $topsrcdir/mail/config/mozconfig
mk_add_options MOZ_OBJDIR=/d/mozilla/mozilla_build
ac_add_options --enable-application=mail
ac_add_options --enable-extensions=default,example
ac_add_options --enable-optimize
ac_add_options --disable-debug
```

Abbildung 3.18: .mozconfig Datei mit Erweiterung

In einem ersten Schritt ist die `.mozconfig` Datei anzupassen. Durch das Hinzufügen von „`example`" hinter „`default`" werden nicht nur die Standarderweiterungen, sondern auch die Beispielerweiterung kompiliert. Damit der Build-Prozess unsere Erweiterung an passender Stelle findet, muss diese, ausgehend vom Hauptverzeichnis des Quellcodes, in das Verzeichnis „`./extensions/example`" abgelegt werden. Die verwendete Ordnerstruktur ist jener in Kapitel *3.1 Einführung Mozilla Programmierung* sehr ähnlich, lediglich das Verzeichnis `components` wird ersetzt und die darin befindlichen Dateien auf die beiden neuen Verzeichnisse `public` und `src` aufgeteilt. In den Ordner `public`, werden alle `.idl` Dateien, in den Ordner `src`, alle Quellcodedateien, wie `.cpp`, `.h` und `.js`, verschoben.

Die Flexibilität des Build-Prozesses beruht darauf, dass jede Erweiterung und auch Thunderbird selbst sogenannte `Makefiles` zur Verfügung stellt, die für Make Anweisungen beinhaltet, wie die Quelldateien kompiliert werden müssen. Um also die Erweiterung mittels des Build-Prozesses zu erstellen, müssen in den Verzeichnissen `example`, `public` und `src` `Makefiles` bereit gestellt werden. Die unten angeführten `Makefiles` müssen für die zu kompilierende Erweiterung angepasst werden. Make ist beim Aufbau eines `Makefiles` sehr genau. Bei Eingabe einer langen Zeichenkette, die sich über mehrere Zeilen erstreckt, muss vor jedem Zeilenumbruch ein „\" eingegeben werden. Am besten ist es, einen Texteditor zu verwenden, der auch Sonderzeichen darstellt, da zum Beispiel ein Leerzeichen nach „\" zu einem Abbruch des Build-Prozesses führt, und derartiges ohne passenden Texteditor nur sehr schwer diagnostiziert werden kann. Im Folgenden wird kurz auf die zu ändernden Zeilen eingegangen.

```
DEPTH     = ../..
topsrcdir = @top_srcdir@
srcdir    = @srcdir@
VPATH     = @srcdir@

include $(DEPTH)/config/autoconf.mk

MODULE    = example
DIRS      = public src

USE_EXTENSION_MANIFEST = 1

XPI_NAME             = example
INSTALL_EXTENSION_ID = {F62DA64F-3337-47bc-9951-9931C01DDADE}
XPI_PKGNAME          = example

DIST_FILES = install.rdf

include $(topsrcdir)/config/rules.mk
```

Abbildung 3.19: Makedatei im Hauptverzeichnis der Beispielerweiterung

`DEPTH = ../..` gibt an, im wievielten Unterverzeichnis, ausgehend vom Hauptverzeichnis des Quellcodes, sich das `Makefile` befindet. In diesem Fall müsste über zwei Verzeichnisse in Richtung Hauptverzeichnis iteriert werden, um in dieses zu gelangen. Einmal nach `extensions` und ein zweites Mal nach `mozilla`.

`MODULE = example` wird verwendet, um Make mitzuteilen, zu welchem Modul die Quelldateien gehören. Alle zu einem Modul gehörenden Dateien werden im Objekt-Verzeichnis gemeinsam abgelegt und sind damit für andere Erweiterungen an einem Punkt vorhanden. Dies ist im nächsten `Makefile` noch von Bedeutung.

`DIRS = public src` gibt an, welche Unterverzeichnisse im Build-Prozess mit berücksichtigt werden sollen.

`XPI_NAME = example` und `XPI_PKGNAME = example` bezeichnen den Namen der Erweiterung.

`INSTALL_EXTENSION_ID = {F62DA64F-3337-47bc-9951-9931C01DDADE}` ist die eindeutige ID einer Erweiterung. Diese sollte aus der `install.rdf` übernommen werden.

```
DEPTH       = ../../..
topsrcdir = @top_srcdir@
srcdir      = @srcdir@
VPATH       = @srcdir@

include $(DEPTH)/config/autoconf.mk

IS_COMPONENT   = 1
MODULE      = example
LIBRARY_NAME   = example

XPI_NAME                = example
INSTALL_EXTENSION_ID = {F62DA64F-3337-47bc-9951-9931C01DDADE}
XPI_PKGNAME           = example

REQUIRES  = $(NULL)

CPPSRCS= \
     examplemodule.cpp \
     nsExample.cpp \
     $(NULL)

EXTRA_COMPONENTS = \
     example.js \
     $(NULL)

#FORCE_STATIC_LIB = 1

include $(topsrcdir)/config/rules.mk

EXTRA_DSO_LDOPTS += \
     $(XPCOM_GLUE_LDOPTS) \
     $(NSPR_LIBS) \
     $(NULL)
```

Abbildung 3.20: Makedatei im src Verzeichnis der Beispielerweiterung

`REQUIRES = $(NULL)` gibt an, dass die Erweiterung keine Dateien anderer Bibliotheken aus Thunderbird oder zusätzlicher Erweiterungen benötigt. Um zu erläutern, wie bei Bedarf die richtige Bibliothek gefunden werden kann, wird nun die Annahme getroffen, dass die obige Erweiterung das Interface `nsIComponentManager` benötigt. Die dazugehörige Interfacebeschreibung liegt in der Datei `nsIComponentManager.idl`, welche sich unter `mozilla/xpcom/components` befindet. In diesem Verzeichnis findet man das Makefile, welches den Build-Prozess mittels „`MODULE = xpcom`“ dazu veranlasst, das in eine Header Datei umgewandelte Interface im Modul `xpcom` abzulegen. Wird `REQUIRES =` um `xpcom` ergänzt, wird der C++ Kompiler durch den Build-Prozess dazu angewiesen, das Verzeichnis des `xpcom` Moduls als zusätzliches Includeverzeichnis zu berücksichtigen.

`CPPSRCS = examplemodule.cpp \` gibt an, welche Dateien kompiliert werden sollen.

EXTRA_COMPONENTS = example.js \ nimmt eine Sonderrolle ein. Da JS eine Interpretersprache ist, müssen Quelldateien von Components, die in JS geschrieben wurden, nicht kompiliert werden. Daher wird der Build-Prozess dazu angewiesen, diese als zusätzliche Dateien ohne Änderung in das .xpi Paket aufzunehmen.

```
DEPTH     = ../../..
topsrcdir = @top_srcdir@
srcdir    = @srcdir@
VPATH     = @srcdir@

include $(DEPTH)/config/autoconf.mk

MODULE        = example
XPIDL_MODULE  = example

XPI_NAME             = example
INSTALL_EXTENSION_ID = {F62DA64F-3337-47bc-9951-9931C01DDADE}
XPI_PKGNAME          = example

XPIDLSRCS = \
     nsIExample.idl \
     $(NULL)

include $(topsrcdir)/config/rules.mk
```

Abbildung 3.21: Makedatei im public Verzeichnis der Beispielerweiterung

XPIDLSRCS = gibt an, welche .idl Dateien automatisch während des Build-Prozesses sowohl in .xpt Dateien als auch in .h Header Dateien umgewandelt werden sollen. Die Header Dateien sind wichtig, da diese von anderen Erweiterungen und innerhalb der eigenen C++ Dateien benutzt werden können.

Sind die oben angeführten Änderungen durchgeführt worden, wird der zusätzlich eingefügte Quellcode der Beispielerweiterung gemeinsam mit Thunderbird kompiliert. Das dabei erstellte example.xpi Paket ist im Anschluss im Objekt-Verzeichnis unter dist\xpi-stage zu finden.

4 Das JackF Projekt

Das primäre Ziel von Thunderbird ist das Bereitstellen eines zuverlässigen Mailclients, weshalb kein besonderes Augenmerk auf einfache Schnittstellen für zusätzliche Erweiterungen gelegt wurde. Eine Konsequenz davon ist, dass es zwar gute Ideen für Erweiterungen gibt, die Realisierung einer solchen jedoch selten vorgenommen wird. Ein Grund dafür ist der hohe Zeitaufwand, den das Einarbeiten in die Programmierung von Erweiterungen erfordert. Dabei macht es keinen Unterschied, ob eine Erweiterung lediglich kleine Teile einer Nachricht überprüfen und daraus resultierende Ergebnisse darstellen möchte, oder komplizierte Prognosen über den Inhalt einer Nachricht aufstellt. Der Grundaufwand ist erheblich und bleibt bei allen Erweiterungen derselbe.

4.1 Überblick

Mit JackF wird versucht, ein entwicklerfreundliches Framework zur einfachen Integration von sicherheitsrelevanten Erweiterungen zu schaffen. Um zwischen normalen Erweiterungen und Erweiterungen, die JackF verwenden, unterscheiden zu können, werden letztere, wie schon erwähnt, Pluggies genannt. In der ersten veröffentlichten Version ist es mit JackF möglich, Ergebnisse durchgeführter Analysetätigkeiten in der Nachrichtenübersicht in einer Spalte auszugeben. Da bereits in der Entwurfsphase auf Erweiterbarkeit geachtet wurde, können weitere Ausgabemöglichkeiten hinzugefügt werden. So ist zum Beispiel die Ausgabe von Analyseergebnissen in den Kopfdaten einer Nachricht oder beim Bewegen des Mauszeigers über eine Nachricht geplant. Ob Analyseergebnisse mit Bildern oder Zeichen dargestellt werden, bleibt dem jeweiligen Pluggy überlassen. Das Framework ermöglicht die Ausgaben in grafischer (✖, ○, ✔), numerischer (1, 2, …) und alphanumerischer (hoch, mittel, niedrig, …) Form und kann unter der Voraussetzung, dass ein Pluggy alle 3 Möglichkeiten unterstützt, vom Anwender frei gewählt werden.

Einen besonders kritischen Faktor von JackF stellte die Performance dar. Während die Analyse einer Nachricht mehrere Sekunden in Anspruch nehmen kann, ist es für die Benutzeroberfläche nicht akzeptabel, länger als 150 ms auf ein Analyseergebnis zu warten [DAMU01]. Viele Analysen können nicht in dieser kurzen Zeit durchgeführt werden, da sie zum Beispiel auf Webservices angewiesen sind, was darin resultiert, dass es scheint, als ob die Benutzeroberfläche einfriert. Aus diesem Grund verwendet JackF zur Analyse im Hintergrund arbeitende Threads mit niedriger Priorität. Die Benutzeroberfläche ist somit während der Analyse

voll funktionsfähig und wird erst nach deren Beendigung über das Analyseergebnis in Kenntnis gesetzt.

Eines der obersten Gebote beim Entwurf von JackF war es, dass der Inhalt einer Nachricht unter keinen Umständen verändert werden darf. Zur weiteren Performancesteigerung erwies es sich jedoch als notwendig, Ergebnisse von durchgeführten Analysen zu speichern und bei erneutem Bedarf wieder abzurufen. Aus diesem Grund wurde in JackF ein von der Nachrichtendatenbank getrennter Datenspeicher integriert. Alle dafür notwendigen Aufgaben übernimmt JackF. Dadurch ist dieser Mechanismus gänzlich vom Pluggy abgekoppelt. Dies erweist sich als besonders wertvoll für Analysen mit langer Laufzeit, bzw. wenn sich das Analyseergebnis zu einem späteren Zeitpunkt mit Gewissheit nicht mehr verändert, wie beispielsweise die Größe einer Nachricht oder die Anzahl der Empfänger.

Um die Entwicklung zusätzlicher Pluggies zu vereinfachen, ist die zu verwendende Schnittstelle besonders einfach gestaltet. Um mit JackF kompatibel zu sein, muss ein Pluggy eine Funktion implementieren und vier Variablen initialisieren. Da die Möglichkeit besteht, den in Thunderbird integrierten Updatemechanismus zu benutzen, ist sichergestellt, dass sowohl JackF als auch dessen Pluggies immer auf dem aktuellen Stand gehalten werden können.

Als Hauptvorteile, die JackF im Vergleich zu Thunderbird den Entwicklern und Nutzern von Erweiterungen bietet, sind zu nennen:

- Pluggys werden durch die Möglichkeit der Ergebnisausgabe in numerischer, alphanumerischer und grafischer Form entlastet.
- Analyseergebnisse werden automatisch und völlig transparent für die Pluggies zwischengespeichert.
- JackF und alle Pluggies werden automatisch durch den Updatemechanismus von Thunderbird aktualisiert.
- Alle Pluggies werden im Konfigurationsmenü von JackF an einem zentralen Punkt eingestellt.
- Es ist möglich, einzelne Pluggies ein- bzw. auszuschalten ohne diese zu deinstallieren.
- JackF bietet Pluggies die Möglichkeit, auf einfache Weise auf Header, Body und Anlage einer Nachricht zuzugreifen.

- Wird eine Nachricht von mehreren Pluggies analysiert, werden ressourcenintensive Aufgaben, wie das Laden einer Anlage von der Festplatte, von JackF nur einmal durchgeführt. Die Ergebnisse stehen allen Pluggies zur Verfügung.
- Das Einbinden neuer Pluggies wird sowohl für Anwender als auch für Pluggy-Entwickler erleichtert.
- Die Analyse von Nachrichten durch Pluggies wird von JackF automatisch von Threads mit niedriger Priorität verrichtet.

4.2 Herausforderung

4.2.1 Einfachheit

„Mache die Dinge so einfach wie möglich, aber nicht einfacher!“ (A. Einstein 1879-1955)

Eine besondere Herausforderung bei der Entwicklung von JackF war es, dessen Benutzung so einfach wie möglich zu gestalten. Die Erfahrung zeigt, dass sich gute Programme nicht durchsetzen, wenn sie für den „normalen“ Anwender ohne besonderes Fachwissen zu schwierig zu handhaben sind. Ein gutes Beispiel dafür stellt das kostenlos erhältliche Betriebssystem Linux dar [Rel03]. Aus diesem Grund ist JackF dafür konzipiert, sowohl Programmierern eine einfache Schnittstelle zu Thunderbird zur Verfügung zu stellen, als auch die von diesen Programmierern entwickelten Pluggies anderen Anwendern zugänglich zu machen. Um beide Ziele zu erreichen, wurden verschiedene Maßnahmen ergriffen, welche im Anschluss näher erläutert werden.

4.2.1.1 Konfiguration an einem zentralen Punkt

Durch die Bündelung aller Pluggies an einer zentralen Stelle ist es möglich, diese gemeinsam ein- bzw. auszuschalten. Es ist auch für den Anwender übersichtlicher, wenn diese nicht mit andern Erweiterungen, wie zum Beispiel einer Erweiterung zum Verschlüsseln von Nachrichten, vermischt werden. Ursprünglich war es geplant, dass Pluggies nicht unter `Tools` → `Add-ons` aufgeführt werden. Auf Grund einer Designentscheidung, welche auf mehr Sicherheit in Thunderbird und Firefox abzielt, ist es leider nicht mehr möglich, dies zu bewerkstelligen. Es müssen alle von einem Anwender installierten Erweiterungen zwingend unter den „normalen“ Erweiterungen angeführt werden, wodurch es nicht mehr möglich ist, dass sich

zum Beispiel Spyware als Erweiterung für den Anwender unsichtbar installiert. Von dieser Designentscheidung unberührt werden im Konfigurationsmenü von JackF lediglich Pluggies aufgelistet. Um die Benutzerakzeptanz weiter zu erhöhen, wurde das Design an jenes von Thunderbird angepasst.

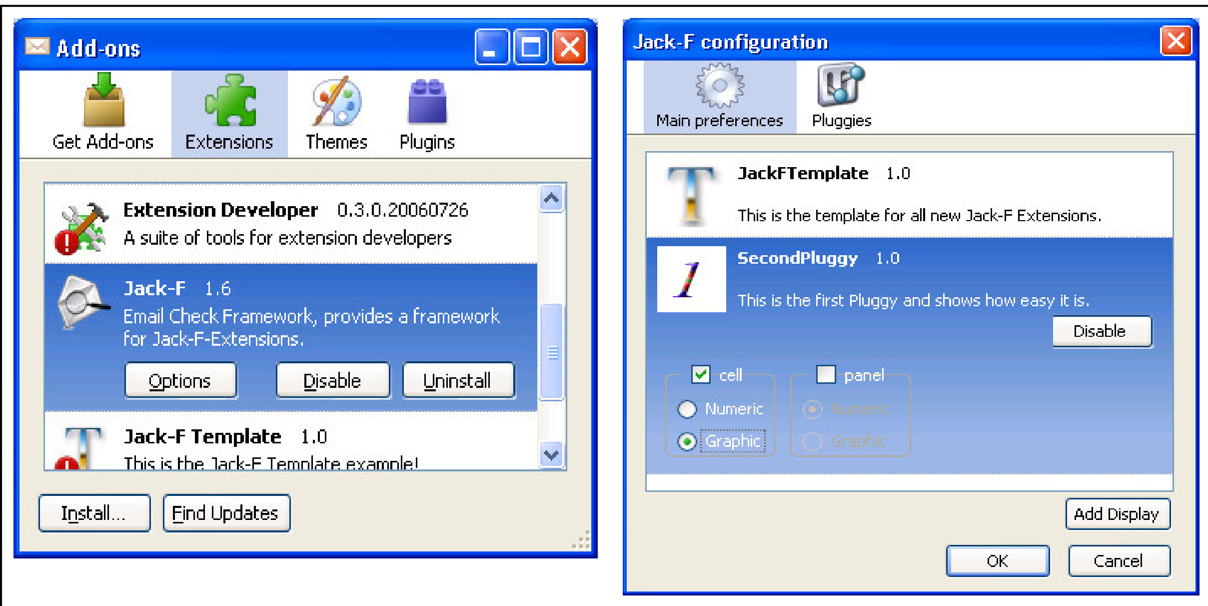

Abbildung 4.1: Add-ons Konfiguration von Thunderbird (li.) und JackF (re.)

4.2.1.2 Übersichtliche Darstellung

Nicht jede Analyse ist in jeder Situation sinnvoll. Daher ist es mit JackF möglich, Analyseergebnisse an unterschiedlichen Orten darzustellen. Wichtig ist, dass die Analyse erst zum Zeitpunkt der Darstellung durchgeführt wird. Daraus ergibt sich, dass ressourcenintensive Analysen, deren Analyseergebnisse nur kurzfristig gültig sind, am besten nicht in einer Spalte in der Nachrichtenübersicht dargestellt werden sollten, auch wenn dies durch das Verwenden von Threads mit niedriger Priorität möglich wäre. Das unten angeführte Beispiel vergleicht den durch den Absender in einer Nachricht eingetragenen Absendezeitpunkt mit dem vom Empfängermailserver eingetragenen Empfangszeitpunkt. Unterscheiden sich diese um mehr als eine Stunde, wird dies durch ein ✘ gekennzeichnet. Da diese Analyse keine aufwendige Berechnung durchführt oder Informationen über einen Webservice abgerufen werden müssen, und das Analyseergebnis lange gültig ist, bieten sich die Ergebnisse dieser Analyse für die Ausgabe in einer Spalte in der Nachrichtenübersicht an.

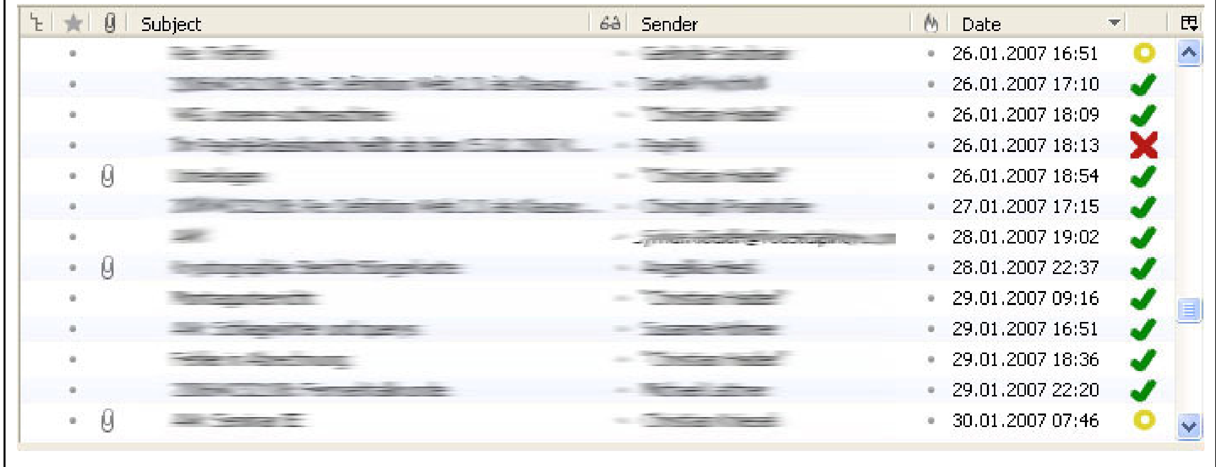

Abbildung 4.2: Analyseergebnis in Spalte ausgegeben

Zeitaufwendigere Analysen, wie zum Beispiel das Prüfen, ob die Nachricht über einen „geblacklisteten" Mailserver versandt wurde, würden für eine große Anzahl an Nachrichten zu viel Zeit in Anspruch nehmen. Daher ist es sinnvoller, die Analyse erst durchzuführen, wenn die Kopfdaten einer Nachricht angezeigt werden, oder wenn der Mauszeiger länger auf einer Nachricht pausiert.

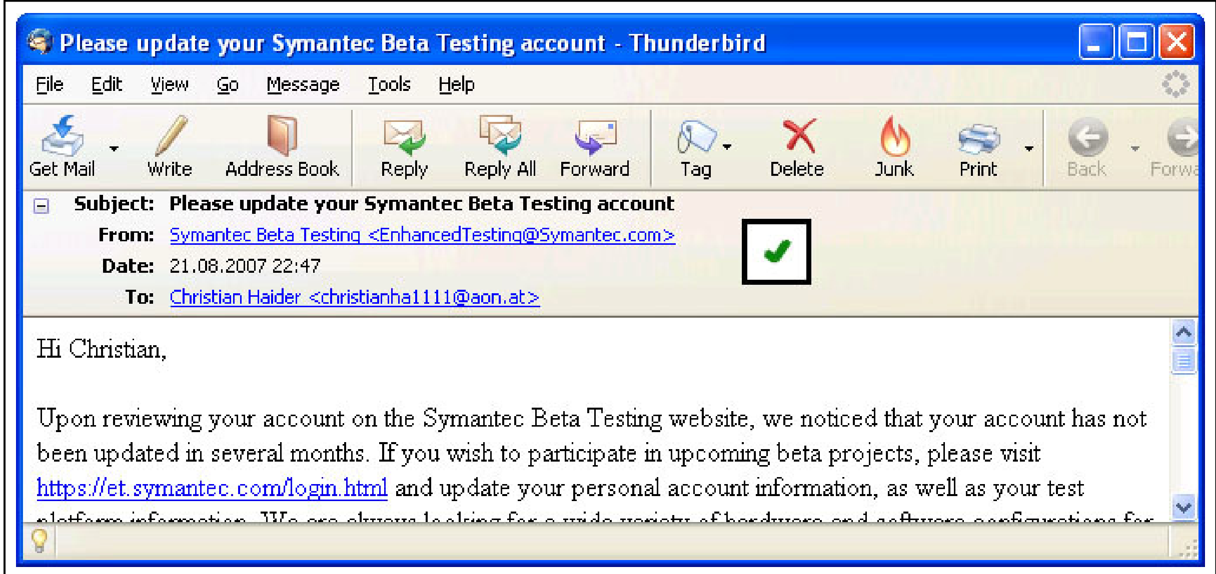

Abbildung 4.3: Analyseergebnis neben Kopfdaten ausgegeben

4.2.1.3 Automatisches Update

Da mit jeder neuen Version von Thunderbird nicht nur Fehler ausgebessert, sondern auch Veränderungen an dessen API vorgenommen werden, ist es notwendig, JackF laufend weiter zu entwickeln. Bereits vor der Veröffentlichung neuer Thunderbird Versionen ist dessen Quellcode ständig in der aktuellen Version verfügbar. Auf diese Weise kann getestet werden,

ob JackF mit diesem kompatibel ist und bereits frühzeitig eine passende Version zur Verfügung gestellt werden. Die Verbreitung einer neuen JackF Version ist durch den Umstand, dass JackF für Thunderbird eine normale Erweiterung ist, unproblematisch. JackF greift auf dieselben Updatemechanismen zurück wie Thunderbird, was bedeutet, dass bei jedem Neustart nach Updates gesucht wird.

Im Zeitraum, in dem JackF entwickelt wurde, wurden zahlreiche Veränderungen an Thunderbird vorgenommen. So war die erste Version von JackF für Thunderbird 1.5 konzipiert. Die derzeitige JackF Version bezieht sich bereits auf den Mitte bis Ende 2008 erscheinenden Thunderbird 3.0. Anzumerken ist, dass die in JackF eingebundenen Pluggies von den jeweiligen Releasewechseln von Thunderbird bisher unberührt blieben, da diese von JackF verwaltet und über die unveränderte Schnittstelle angesprochen werden. Sollte ein Pluggy einen Fehler aufweisen, kann der Pluggy-Entwickler diesen mit dem automatischen Updatemechanismus von Thunderbird beheben und eine neue Version bereitstellen.

4.2.1.4 Einfach zu verwendende Schnittstellen

Wie bereits in Kapitel *4.1 Überblick* erwähnt, ist der Aufwand, welcher für die Einbindung einfacher Erweiterungen betrieben werden muss, erheblich, was dazu führt, dass viele gute Ideen an der praktischen Umsetzung scheitern. Aus diesem Grund beschränkt sich JackF auf 4 Interfaces:

- *nsIJackFExtension:* Beinhaltet Variablen für den Namen des Pluggys, eine Beschreibung, ein Logo, die Version, ein Time-out und den gewünschten Zeitraum, in dem Analyseergebnisse zwischengespeichert werden sollen. Um JackF die Verwaltung von Pluggies zu ermöglichen, ist das Interface `nsIJackFExtension` von allen Pluggies zwingend zu implementieren.
- *nsIJackFNumericExtension:* Ist jenes Interface, welches für numerische Analyseergebnisse verwendet wird. Die einzige enthaltene Funktion ist `analyseNumeric(…)`.
- *nsIJackFAlphanumericExtension:* Sorgt dafür, dass Analysen mit alphanumerischen Analyseergebnissen durchgeführt werden können. Die enthaltene Funktion `analyseAlphanumeric(…)` ist zwingend zu implementieren. Die zweite Funktion `convertToAlphanumeric(…)` ist optional und ermöglicht das Umrechnen eines numerischen Analyseergebnisses in ein alphanumerisches. Dadurch ist es möglich, gespei-

cherte numerische Analyseergebnisse, wie 1/2/.../10, in alphanumerische, hoch/mittel/niedrige, umzurechnen, ohne eine erneute Analyse der Nachricht durchzuführen.

- *nsIJackFGraphicExtension:* Das Interface und die darin enthaltene Funktion `analyseGraphic(…)` wird für Analysen mit grafischen Analyseergebnissen verwendet. Die optionale Funktion `convertToGraphic(…)` erfüllt einen ähnlichen Zweck wie jene im Interface `nsIJackFAlphanumericExtension`, nur dass sie ein numerisches Analyseergebnis in ein grafisches umwandelt.

nsIJackFExtension

+GetName(ein aName : nsACString &) : nsresult
+GetDescription(ein aDescription : nsACString &) : nsresult
+GetLogo(ein aLogo : nsACString &) : nsresult
+GetVersion(ein aVersion : nsACString &) : nsresult
+GetTimeout(ein aTimeout : PRInt32) : nsresult*
+GetCachetime(ein aCachetime : PRInt32) : nsresult*
+GetPreferedDisplays(ein length : PRUint32, ein displays : char***) : nsresult*

nsIJackFNumericExtension

+AnalyseNumeric(ein msg : nsIJackFMsg, ein _retval : PRInt32*) : nsresult*

nsIJackFAlphanumericExtension

+AnalyseAlphanumeric(ein msg : nsIJackFMsg, ein _retval : nsACString &) : nsresult*
+ConvertToAlphanumeric(ein score : PRInt32, ein _retval : nsACString &) : nsresult

nsIJackFGraphicExtension

+AnalyseGraphic(ein msg : nsIJackFMsg, ein _retval : nsACString &) : nsresult*
+ConvertToGraphic(ein score : PRInt32, ein _retval : nsACString &) : nsresult

Abbildung 4.4: JackF Pluggy-Interfaces

Um die einzelnen Pluggies unterscheiden und die in Thunderbird integrierten Installations- und Updatemechanismen verwenden zu können, werden alle Pluggies wie herkömmliche Erweiterungen installiert. Das Erstellen eines Pluggies wird in Kapitel 4.6 *Schreiben eines Pluggies – Entwicklerhandbuch* für Entwickler Schritt für Schritt erklärt und ist somit für jedermann leicht verständlich.

Ein Pluggy kann Analyseergebnisse wahlweise in numerischer, alphanumerischer und grafischer Form bereitstellen und muss dazu die dafür erforderlichen Interfaces implementieren.

Da in vielen Fällen eine Umrechnung von numerischen Analyseergebnissen in alphanumerische oder grafische Analyseergebnisse möglich ist, sollten die beiden optionalen Funktionen zum Umrechnen zusätzlich implementiert werden. Das einfachste mit JackF kompatible Pluggy, welches ausschließlich für numerische Analysen bestimmt ist, muss die Funktion `analyseNumeric(…)` implementieren und die zur Identifikation benötigen Variablen aus `nsIJackFExtension` setzen.

Jeder der drei Analysefunktionen wird die gesamte Nachricht übergeben. Dies ermöglicht den Pluggy Zugriff und damit eine Analyse des Headers, Bodys sowie auch aller Anhänge einer Nachricht. Dies ist wichtig, damit JackF jede Art von Analyse ermöglicht, und dennoch die Schnittstelle einfach und flexibel bleibt. Das in Abbildung *2.11: Spamness Beispiel* gezeigte Beispiel würde mit einem Pluggy realisiert 22 Codezeilen benötigen. Dies ist möglich, da JackF alle Aufgaben wie Anzeigen der Analyseergebnisse, Verwalten der Threads, Speichern der Analyseergebnisse und viele andere mehr übernimmt.

4.2.2 Performance

Für die Benutzbarkeit von JackF ist es entscheidend, dass der Anwender von den im Hintergrund ablaufenden Analysen nicht beeinträchtigt wird. Dazu ist es notwendig, dass JackF mit möglichst niedriger Latenz auf alle gestellten Anfragen reagiert. [DaMu01] geht davon aus, dass eine Antwortzeit von 150 ms für eine Benutzeroberfläche ohne Akzeptanzeinbuße toleriert wird. In Bezug auf JackF wurde daher das Ziel gesetzt, Anfragen unter 2 ms zu beantworten. Dies ist wichtig, damit JackF Analyseergebnisse in einer Spalte in der Nachrichtenübersicht darstellen kann. Wenn 20 Nachrichten mit jeweils drei Pluggies, also 60 Anfragen, unter 150 ms darzustellen sind, muss jede dieser Anfragen im Durchschnitt in 2,5 ms beantwortet werden. Tabelle 4.1: Performancevergleich zweier JackF Versionen zeigt die Resultate einer Performancemessung einer in einem früheren Stadium befindlichen JackF Version und JackF in der derzeitigen Version. Erstere war für Thunderbird 2.0.0.6 gedacht und gänzlich in JS implementiert. Die aktuelle Version ist für Thunderbird 3.0 geeignet und wird bei dessen Erscheinen verfügbar sein. Teile, welche die Geschwindigkeit beeinflussen, sind in C++ implementiert. Zur Messung wurde ein Nachrichtenarchiv mit 5.169 Nachrichten genutzt. Das verwendete Referenzgerät ist ein handelsüblicher Computer mit einem 2.1 GHz Prozessor und 1GB Arbeitsspeicher, auf dem das Betriebssystem Windows XP Professional installiert ist. Die verwendete Einheit der Zeitangaben in der Tabelle sind Sekunden. Die in Klammern ge-

schriebenen Angaben repräsentieren den Hauptspeicherbedarf nach beendeter Messung in Megabyte.

Tabelle 4.1: Performancevergleich zweier JackF Versionen

	Frühe Version				Aktuelle Version			
Versuch	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Blättern ohne JackF	15 (31)	15 (30)	15 (31)	15 (31)	15 (43)	15 (40)	15 (40)	15 (40)
Blättern über alle Testnachrichten exklusive Analyse der Nachrichten	26 (67)	26 (70)	27 (67)	26 (67)	20 (49)	21 (48)	22 (48)	21 (49)
Blättern, wenn alle Anfragen zwischengespeichert sind	24 (43)	26 (42)	24 (42)	25 (42)	17 (43)	16 (42)	16 (42)	16 (42)
Blättern über alle Testnachrichten inklusive Analyse der Nachrichten	87 (69)	88 (70)	90 (67)	87 (67)	21 (49)	23 (46)	22 (47)	23 (47)

Wie aus der obigen Abbildung ersichtlich, benötigt Thunderbird ohne JackF 15 Sekunden zum Durchblättern der Nachrichtenübersicht mit 5.169 Nachrichten mit der Bild-Aufwärts- bzw. Bild-Abwärts-Taste. Ist die aktuelle JackF Version installiert, werden, wenn keine Analyseergebnisse zwischengespeichert sind, max. 7 Sekunden zusätzlich benötigt. Dies ergibt eine Antwortzeit von durchschnittlich 1,35 ms. Im Falle, dass alle Analyseergebnisse zwischengespeichert sind, kann die benötigte Antwortzeit auf ca. 0,39 ms reduziert werden. Zu berücksichtigen ist, dass die oben angeführten Zeiten nicht nur die Bearbeitung von JackF inkludieren sondern auch die zusätzlich von Thunderbird zum Anzeigen benötigte Zeitspanne enthalten. Eine genaue Zeitmessung mit 140.000 Analyseanfragen ergab, dass JackF am Referenzgerät durchschnittlich 1,04 ms zur Beantwortung einer nicht zwischengespeicherten Analyseanfrage und 0,06 ms bei einer zwischengespeicherten Analyseanfrage benötigt. Dies unterbietet das geforderte Ziel von 2,5 ms bzw. das gesetzte Ziel von 2 ms und ist somit für die Benutzerakzeptanz ausreichend. Um die niedrigen Antwortzeiten zu bewerkstelligen, wurde in JackF eine Vielzahl an performancesteigernden Mechanismen eingebaut, welche nachfolgend näher beschrieben wird.

4.2.2.1 Threading

Wird die Analyse einer Nachricht angefordert, wird entweder ein bereits gespeichertes Analyseergebnis oder eine Mitteilung darüber, dass noch kein Analyseergebnis vorliegt, angezeigt. Durch diese Vorgehensweise ist es möglich, die Benutzeroberfläche während der Analyse

nicht unnötig warten zu lassen. JackF speichert alle unbeantworteten Anfragen und benachrichtigt nach erfolgreicher Analyse das wartende Element der Benutzeroberfläche.

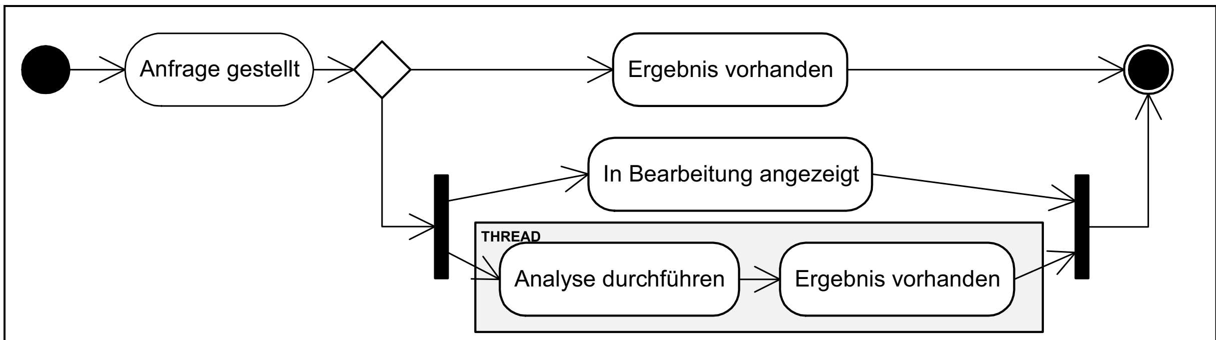

Abbildung 4.5: Schematische Darstellung einer Anfrage

Wie die obige Abbildung zeigt, wird die eigentliche Analyse einer Nachricht von Threads durchgeführt. Diese haben die niedrigste, am jeweiligen System verfügbare, Priorität. Das Erzeugen eines Threads ist ein zeitaufwändiger und ressourcenintensiver Prozess. Aus diesem Grund verwendet JackF einen Threadpool. Die Größe dieses Pools kann in der Benutzeroberfläche von JackF frei gewählt werden, wobei der Standardwert 10 gleichzeitig laufende Threads beträgt. In den meisten heute eingesetzten Systemen verrichten zwei Prozessorkerne ihre Arbeit, daraus resultieren im Durchschnitt 5 Threads pro Prozessorkern.

Ist die höchstzulässige Anzahl an Threads erreicht, speichert JackF alle abzuarbeitenden Anfragen in einer Warteschlange. Es kann davon ausgegangen werden, dass die zuletzt gestellten Anfragen für den Anwender die höchste Priorität haben, weshalb die Warteschlange nach dem Last-In-First-Out Prinzip abgearbeitet wird. Hat ein Thread seine Arbeit beendet, und ist die Warteschlange leer, wird der Thread beendet und erst bei Bedarf neu erstellt.

4.2.2.2 Terminieren von Threads

Jedes Pluggy hat die Möglichkeit anzugeben, wie lange es höchstens für eine Analyse benötigt. Wird diese Dauer nicht angegeben, werden von JackF 50 ms veranschlagt. Wird eine Analyse in dem für sie eingeräumten Zeitraum nicht erfolgreich beendet, terminiert JackF deren Thread. Dadurch wird verhindert, dass fehlerhaft programmierte Pluggies das Framework blockieren, und eine DOS Situation auftritt. Für den Fall, dass ein Pluggy mehrere Male hintereinander nicht erfolgreich beendet werden kann, könnte eine Erweiterung von JackF dahingehend erfolgen, dass dieses Pluggy bis zum nächsten Start von Thunderbird deaktiviert wird. Mit der Time-out-Einstellung gibt JackF den Entwicklern von Pluggies die Freiheit, die

benötigte Zeitspanne frei zu wählen. Dies ist notwendig, da es Pluggies gibt, die analysebedingt von Haus aus eine längere Laufzeit haben.

4.2.2.3 Implementieren der Kernfunktionen in C++

Zu Beginn war das gesamte JackF-Framework in JS implementiert. Dies hatte neben einer schnelleren Entwicklungsphase den Vorteil, dass JackF uneingeschränkt plattformunabhängig entwickelt werden konnte. Eine Portierung der Kernfunktionen von JS nach C++ war mit dem Releasewechsel von Thunderbird 2.0.0.6 nach Thunderbird 3.0a1pre notwendig, da einige wichtige Funktionen der API nicht mehr in JS verwendbar waren. So war es zum Beispiel nicht mehr möglich, Threads für einen bestimmten Zeitraum pausieren zu lassen. Um dennoch ein gewisses Maß an Plattformunabhängigkeit zu wahren, wurde JackF in den Build-Prozess von Thunderbird eingebunden. Die Umstellung auf C++ hatte nicht nur den Vorteil, dass alle Funktionen der von Thunderbird zur Verfügung gestellten API uneingeschränkt benutzt werden konnten, sondern auch, dass eine höhere Geschwindigkeit erzielt werden konnte, was die Benutzerakzeptanz fördert. Das vollständige Blättern und Analysieren mit einem Musterpluggy, welches eine einfache Analyse des Absendezeitpunktes durchführte, nahm bei einer Nachrichtenübersicht mit 5.169 Nachrichten mit der reinen JS Implementierung zwischen 87 und 90 Sekunden in Anspruch. Die benötigte Dauer konnte mit Hilfe von C++ auf ca. 23 Sekunden reduziert werden. Siehe dazu Tabelle *4.1: Performancevergleich zweier JackF Versionen*. Dies lässt sich zum Großteil auf drei Begebenheiten zurückführen:

1. Thunderbird unterstützt kein echtes Multithreading für JS. Auch wenn es den Anschein erweckt, dass mehrere Threads gleichzeitig in der JS Engine abgearbeitet werden, wird dies lediglich simuliert. Für den Einsatz von in JS geschriebenen Pluggies bedeutet dieser Umstand, dass sie zwar nicht parallel bearbeitet werden, die Benutzeroberfläche jedoch trotzdem nicht beeinträchtigt wird.
2. Viele der von JackF benutzen Components sind in C++ geschrieben. Werden diese in JS benutzt, muss XPConnect deren Interoperabilität bewerkstelligen, was einen zusätzlichen Overhead darstellt.
3. JS ist im Gegensatz zu C++ eine Interpretersprache. Dies hat zwar den Vorteil einer quasi Plattformunabhängigkeit, gerade dadurch ist es jedoch im Vergleich zu C++ langsamer. Die in C++ geschriebenen Components werden bereits vor der Ausführung kompiliert und müssen damit nicht mehr interpretiert werden.

4.2.2.4 Gemeinsame Aufgaben kombinieren

JackF versucht alle Aufgaben, die mehrmals für verschiedene Analysen ein und derselben Nachricht benötigt werden, nur einmal auszuführen. Als Beispiel dafür kann angeführt werden, dass jede Nachricht mit einer eindeutigen Identifikationsnummer versehen sein sollte. Diese Identifikationsnummer sollte bereits der Mailserver beim Versandt einer Nachricht vergeben. Leider wurde in einem Thunderbird Release verabsäumt, für Nachrichten, welche keine Identifikationsnummer vom Mailserver erhalten haben, eine Identifikationsnummer zu berechnen. Dadurch kann es vorkommen, dass sich Nachrichten im Datenspeicher befinden, welche nicht eindeutig identifiziert werden können. Für eine solche Nachricht wird von JackF eine MD5 Prüfsumme der gesamten Nachricht berechnet und diese als Identifikationsnummer verwendet. Eine weitere ressourcenintensive Aufgabe ist das Laden von Nachrichtenanhängen, weshalb diese bei Bedarf in den Hauptspeicher geladen werden und somit mehreren Pluggies gleichzeitig zur Verfügung stehen.

4.2.3 Plattformunabhängig

Da Thunderbird für eine Vielzahl von Betriebsystemen verfügbar ist, sollte auch JackF für diese erhältlich sein. Wie bereits erwähnt sollte dies erreicht werden, indem JackF gänzlich in JS implementiert wird. Auf Grund von internen Änderungen im Kern von Thunderbird einerseits und Performanceproblemen andererseits wurde dieses Ziel zu Gunsten von Stabilität und Geschwindigkeit gänzlich dahingehend abgeändert, dass JackF in den Build-Prozess von Thunderbird eingebunden werden können muss. Dadurch ist es ohne Aufwand möglich, JackF auf jedem Betriebssystem zu kompilieren, für welches auch Thunderbird verfügbar ist. Das beim Build-Prozess erzeugte Binary kann wie Thunderbird selbst für Prozessoren und Betriebssysteme optimiert werden, was zusätzlich zu einem Performancegewinn führt. Möchte ein Anwender JackF installieren, muss er entsprechend seines Betriebssystems das dazu passende Installationspaket auswählen.

Um die Entwicklung von Pluggies auch Personen zugänglich zu machen, die C++ nicht beherrschen, besteht natürlich weiterhin die Möglichkeit, Pluggies in JS zu entwickeln. Dabei gewährleistet XPConnect die Interoperabilität zwischen JS und C++. Der Overhead, der bei einmaligem Aufruf pro Analyse einer Nachricht auftritt, ist zu klein, als dass man Performanceeinbußen erkennen könnte. Problematischer ist es, wenn innerhalb eines Pluggys eine Vielzahl von aufwändigen Berechnungen durchgeführt wird. In diesem Fall bietet es sich an, die

Analyseergebnisse so lange wie möglich in JackFs integrierter Datenbank abzuspeichern, um diese Analyse bei erneutem Bedarf nicht nochmals durchführen zu müssen.

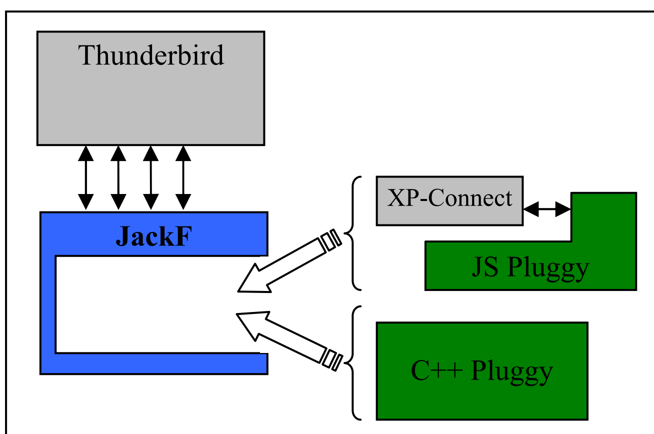

Abbildung 4.6: Schematische Darstellung von JackF

Mit der oben beschriebenen Vorgehensweise ist ein guter Kompromiss zwischen „Plattformunabhängigkeit“ oder zumindest Portierbarkeit auf andere Plattformen und Geschwindigkeit und Stabilität erzielt worden.

4.2.4 Zwischenspeichern von Analyseergebnissen

JackF soll Entwicklern das Erstellen von Erweiterungen vereinfachen. Daher werden performancesteigernde Maßnahmen bereits in JackF eingebaut. Ein Großteil der Analysen, die erhebliche Ressourcen in Anspruch nehmen oder designbedingt mehr Zeit benötigen, muss nur einmal durchgeführt werden. In einer von [BNSO08] durchgeführten Umfrage ist Performance für 13,6 Prozent der Befragten ein wichtiger Indikator für Softwarequalität. Muss eine Analyse nur einmal durchführt werden, ist es nicht mehr notwendig, Pluggies nach Ihrer Geschwindigkeit zu beurteilen sondern danach, ob diese fehlerfrei und wartungsfreundlich programmiert wurden. Dies soll Programmierer dazu anhalten, mehr Zeit damit zu verbringen, „fehlerfreien“ Code zu schreiben, als einen Großteil der Zeit damit zu vergeuden, besonders schnelle Pluggies mit ausgefallenen, hoch performanten Algorithmen zu entwerfen. Durch das Zwischenspeichern von Analyseergebnissen geht JackF noch einen Schritt weiter in Richtung qualitativer Software. Die von [NSDO08] Befragten gaben weiters an, dass ein Programm wartungsfreundlich programmiert sein sollte. Dies wird durch die Verkürzung des Programm-Codes eines Pluggies zusätzlich erreicht.

Das folgende Beispiel soll das enorme Einsparungspotenzial, welches das Zwischenspeichern von Analyseergebnissen beinhaltet, veranschaulichen. Angenommen ein durchschnittlicher Anwender erhält ungefähr 20 Nachrichten pro Tag und hat 40 Nachrichten im Posteingang. Diesen öffnet er alle 15 Minuten, um neue Nachrichten zu lesen. Innerhalb von acht Stunden wäre es somit notwendig, 40*4*8=1.280 Analysen durchzuführen. Wenn jede Analyse innerhalb von 50ms durchgeführt werden würde, würde dies insgesamt 64 Sekunden in Anspruch nehmen. Das Abfragen eines gespeicherten Analyseergebnisses benötigt in der derzeitigen Implementierung 0,06 ms. Da auch die 20 neuen Nachrichten einmal analysiert werden müssen, ergibt sich ein Gesamtaufwand von: (1.280*0,06+20*50)/1.000 ~ 1,08 Sekunde.

Das Zwischenspeichern von Analyseergebnissen ist auch deshalb notwendig, da JackF Analyseergebnisse in einer Spalte in der Nachrichtenübersicht darstellen können soll. Die Nachrichtenübersicht von Thunderbird kann mit einer Tabelle verglichen werden, in welcher jede Zeile eine Nachricht beinhaltet. Thunderbird hat die Eigenschaft, dass der Inhalt jeder Zelle beim Scrollen mit der Maus oder den Cursortasten aktualisiert wird, auch wenn sich die Zelle lediglich um eine Position nach unten oder oben verschiebt. Beim Bewegen der Maus über eine Nachricht, wird in unregelmäßigen Abständen, in der Regel etwa alle drei Pixel, eine Aktualisierung der betroffenen Zeile vorgenommen. Beim Scrollen über die als Testfälle verwendeten 5.169 Nachrichten stellt Thunderbird im Durchschnitt 72.000 Anfragen an JackF. Bei einer erneuten Analyse jeder dieser Anfragen würde JackF 60 Minuten im Gegensatz zu 4,32 Sekunden benötigen.

JackF verwendet zwar bei der Analyse von Nachrichten Threads mit niedrigster Priorität, zur Darstellung der Analyseergebnisse muss dennoch der `MainThread` von Thunderbird verwendet werden. Daher ist es notwendig, dass die einzelnen Threads zur Kommunikation Events an den `MainThread` senden. Das Senden eines Events ist eine ressourcenintensive Aufgabe, welche beim Anzeigen von zwischengespeicherten Analyseergebnissen entfällt, was zusätzlich eine Performancesteigerung bedeutet.

An der in JackF enthaltenen Datenbank wurden seit JackF Version 1.0 entscheidende Veränderungen vorgenommen. Während die Datenbank zu Beginn als RDF Datei implementiert war, und Änderungen von jedem Thread aus vorgenommen werden konnten, ist dies wegen Stabilitäts- und Konsistenzproblemen nun ausschließlich vom `MainThread` aus möglich. Dies wirkt sich nicht negativ auf die Performance von JackF aus, da ohnehin jeder Thread die

Ausgabe eines Analyseergebnisses ausschließlich über den `MainThread` veranlassen kann. Im Zuge dessen wird das Analyseergebnis in die Datenbank gespeichert. Es wurde nicht nur der Zugriff auf die Datenbank sondern auch die Datenbank selbst geändert. Es wird nicht mehr eine RDF Datei verwendet sondern die in Thunderbird integrierte, frei erhältliche SQLite Datenbank. Diese speichert die Daten sehr kompakt, so benötigt diese zum Speichern der 5.169 Ergebnisse lediglich 1.420 KB Speicherplatz im Vergleich zu der RDF Datenhaltung mit 3.120 KB. Da bereits bei der Planung von JackF davon ausgegangen wurde, dass die RDF Datenbank zu einem späteren Zeitpunkt gegen eine performantere Datenbank ausgetauscht wird, war dies bereits im Design berücksichtigt und konnte ohne eine Designänderung vorgenommen werden.

4.3 Benutzerhandbuch

In diesem Kapitel wird erläutert, wie JackF und Pluggies installiert und konfiguriert werden. Es wird nicht darauf eingegangen, wie ein Pluggy für JackF geschrieben wird. Dafür wird auf Kapitel *4.6 Schreiben eines Pluggies – Entwicklerhandbuch* verwiesen.

4.3.1 Installation

Das Installieren von JackF kann vorübergehend nur auf eine Weise erfolgen. Dazu ist es notwendig, das Installationspaket mit dem Namen `jackf.xpi` per Drag & Drop in das `Add-ons` Fenster zu ziehen. Da JackF plattformabhängig kompiliert wird, muss zwischen den Installationspaketen für Windows, Linux und MAC OSX unterschieden werden. Um zum `Add-ons` Fenster zu gelangen, muss der Anwender in der Menüleiste `Tools` → `Add-ons` öffnen. Da JackF noch nicht in die „offizielle" Liste empfohlener Erweiterungen[a] aufgenommen wurde, ist die zweite Installationsvariante für Erweiterungen, bei welcher der Anwender im `Add-ons` Fenster den Karteireiter `Get Add-ons` verwendet, vorübergehend noch nicht anwendbar. Nach dem von Thunderbird vorgeschlagenen Neustart ist JackF installiert und kann konfiguriert werden.

[a] https://addons.mozilla.org/en-US/thunderbird

4.3.2 Konfiguration

Um zum Konfigurationsmenü einer Erweiterung zu gelangen, ist es in Thunderbird üblich, diese im `Add-ons` Fenster auszuwählen und mittels Drücken des mit `Options` beschrifteten Buttons das Konfigurationsmenü zu öffnen. Das Konfigurationsmenü von JackF ist jenem von Thunderbird angepasst. Einen Vergleich dieser sieht man in Abbildung *4.1: Add-ons Konfiguration von Thunderbird (li.) und JackF (re.)*.

Die Konfiguration von JackF erfolgt in zwei Karteireitern. Der erste Karteireiter mit dem Namen `Main preferences` ermöglicht es einzustellen, ob und wie ein Pluggy eine Nachricht analysiert und wo dessen Analyseergebnis ausgegeben wird. Die Konfiguration der Ausgabe erfolgt in drei Schritten.

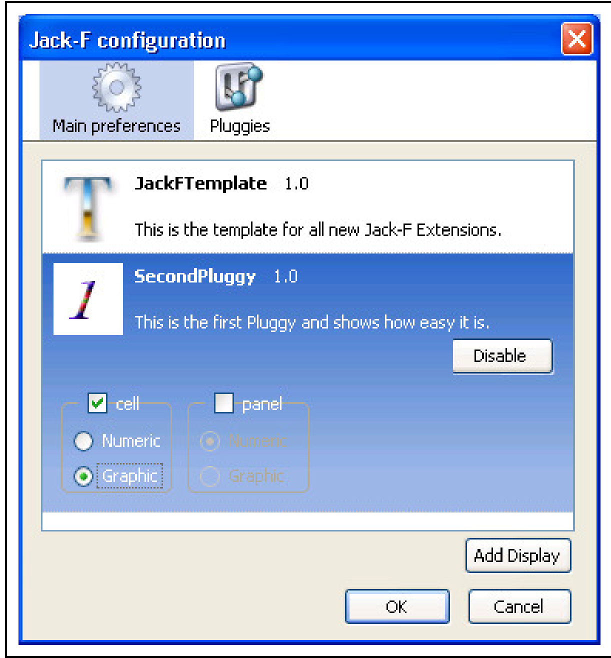

Abbildung 4.7: Konfigurationsmenü JackF

In Schritt eins ist ein Display durch Drücken des `Add Display` Buttons anzulegen. In dem sich öffnenden Eingabefenster sind ein Name, eine Beschreibung und eine Versionsnummer anzugeben, diese repräsentieren später das Display im Karteireiter `Main preferences`. Im zweiten Schritt muss für das erstellte Display gewählt werden, welches Pluggy sein Ergebnis darin darstellen soll. Um dies zu bewerkstelligen, ist das Display auszuwählen und mittels Drücken auf den `Configure` Button zu öffnen. Wie die untere Abbildung zeigt, werden dazu alle installierten Pluggies aufgelistet. Das äußerst linke Häkchen gibt an, ob ein Pluggy für ein Display aktiviert wurde. Durch Klicken auf dieses kann das Pluggy ein- bzw. ausgeschaltet

werden. Die drei daneben stehenden Häkchen geben an, welche Analyseformen ein Pluggy unterstützt.

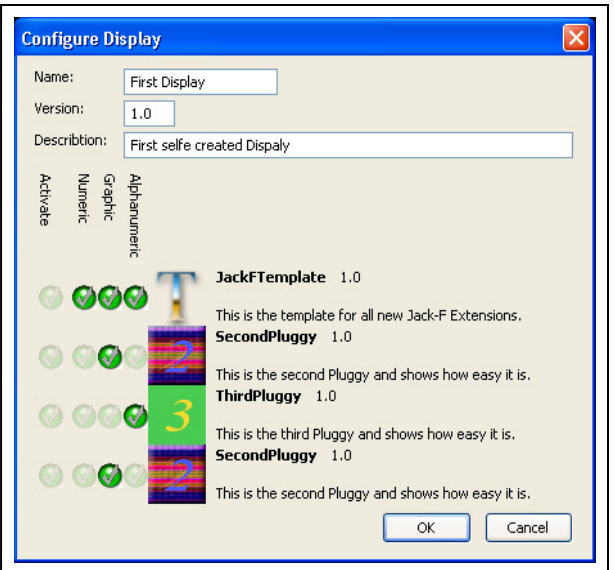

Abbildung 4.8: Konfiguration eines Displays

Der letzte Schritt ist es, unter `Main preferences` zu wählen, wo und in welcher Form das Display ausgegeben werden soll. Dabei werden nur jene Ausgabeformen angeboten, welche vom gewählten Pluggy tatsächlich unterstützt werden. Ein angelegtes Display kann, wenn dieses nicht länger benötigt wird, durch Drücken des `Delete` Buttons aus der Konfiguration gelöscht werden.

Um die Konfiguration von JackF zu vereinfachen, wird automatisch für jedes neu installierte Pluggy ein Display angelegt und der Name, die Beschreibung und die Version des Pluggys für das Display übernommen. Diese automatisch angelegten Displays können durch den Anwender weder gelöscht noch geändert werden. Es ist lediglich möglich, diese durch Drücken des `Disable` Buttons zu deaktivieren. Es mag den Anschein erwecken, dass die Möglichkeit Displays zu erzeugen nicht benötigt wird, dem ist aber nicht der Fall, da durch Anlegen eines zusätzlichen Displays dem Anwender ermöglicht wird, Analyseergebnisse eines Pluggys gleichzeitig in numerischer als auch alphanumerischer Form auszugeben. Erklärtes Ziel von JackF in einer Folgeversion ist es, dass Kombinieren von Pluggies zu erlauben. Dazu ist es notwendig, mehrere Pluggies in einem Display zu aktivieren. Auch dazu werden zusätzliche Displays benötigt.

Der zweite Karteireiter mit dem Namen `Pluggies` ermöglicht die Konfiguration der Pluggies und ist für erfahrene Anwender gedacht. Änderungen an der Konfiguration eines Pluggies

betreffen alle Displays, in welchem dieses dargestellt wird. Die erste Einstellung in diesem Karteireiter betrifft die Anzahl der gleichzeitig im Hintergrund durchgeführten Analysen. Diese ist standardmäßig auf 10 eingestellt und kann entsprechend der Leistung des Computers angepasst werden. Durch Drücken des `Configure` Buttons öffnet sich ein Konfigurationsmenü, in dem sowohl ein Time-out, welcher angibt, wie lange ein Pluggy für eine Analyse benötigen darf, als auch ein Zeitraum, in dem ein Analyseergebnis für erneutes Anzeigen zwischengespeichert wird, einzustellen ist.

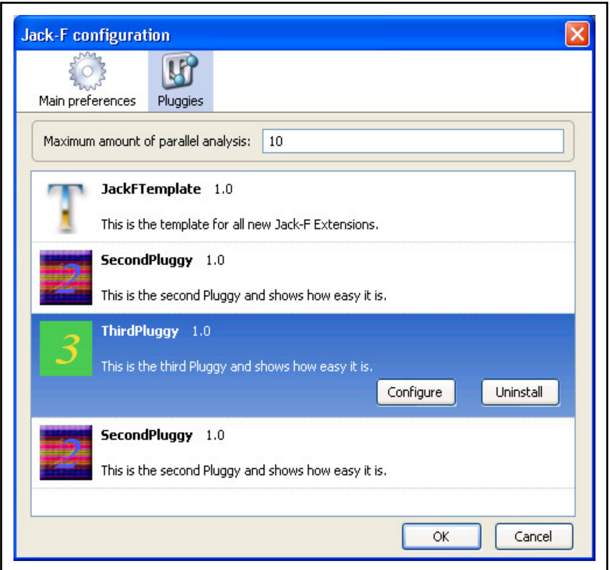

Abbildung 4.9: Konfiguration eines Pluggies

4.4 Architektur

Dieses Kapitel ist in die Teile Display Klassen, Datenbank Klassen, Konfigurationsklassen, JackF Kernstück, Worker Klassen, Hilfsklassen und Zusätzliche Diagramme gegliedert. Jedes dieser Unterkapitel wird schrittweise verfeinert. Mit Klassendiagrammen wird der grundlegende Aufbau beschrieben. Weiters werden die Wechselwirkungen zwischen den Klassen und deren interne Zustände durch Zustandsdiagramme und Sequenzdiagramme näher erläutert. Dadurch soll aufgezeigt werden, warum sich das gewählte Design besonders durch hohe Flexibilität auszeichnet.

Wie in XPCOM üblich müssen alle Components vom Interface `nsISupports` abgeleitet werden. Der Leser soll darauf hingewiesen werden, dass nicht aus Unwissenheit sondern zur Wahrung der Übersichtlichkeit darauf verzichtet wurde, Interfaces und Klassen, welche diese implementieren, darzustellen. Auch wurde davon abgesehen, grafisch zwischen Klassen und Interfaces zu unterscheiden. Bei allen Objekten, die mit „`nsI`" beginnen, handelt es sich um

Interfaces. Um sichtbar zu machen, dass ein Interface von einer Klasse implementiert wird, wird das Interface durch ein C in der rechten oberen Ecke gekennzeichnet. Um anzuzeigen, dass es sich dabei auch um eine XPCOM Component handelt, wird das Interface zusätzlich mit einen X versehen. XPCOM bietet dem Programmierer einige angenehme Hilfsfunktionen wie etwa das Referencecounting und das Freigeben nicht mehr benötigten Arbeitsspeichers, die das Programmieren vereinfachen und dafür sorgen, dass weniger Fehler in der Implementierung entstehen. Um diese Hilfsfunktionen verwenden zu können und die Wiederverwendbarkeit zu steigern, wurde der Großteil der von JackF implementierten Klassen als XPCOM Components realisiert. Natürlich ist mit der Benutzung von Components ein Overhead verbunden. Wird aus Effizienzgründen darauf verzichtet, eine Component zu benutzen, wird an entsprechender Stelle gesondert darauf hingewiesen. Um zu unterscheiden, welche der Klassen in C++ und welche in JS implementiert sind, werden diese jeweils durch C++ oder JS gekennzeichnet.

Einige der Bezeichnungen beschriebener Objekte beinhalten den Ausdruck „`manager`“. Das bedeutet, dass es sich um einen Service handelt. Ein Service, der von unterschiedlichen Threads aus benutzt werden können soll, bedarf besonderer Vorkehrungen. In einem derartigen Fall muss der Service als Threadsafe gekennzeichnet und gänzlich „reentrant“ implementiert werden. Dies bedeutet, dass das Ausführen einer Funktion dieses Services zu jedem Zeitpunkt unterbrochen, und eine andere Funktion gestartet werden kann, ohne dass der interne Zustand dieses Services ungültig wird oder nicht vorhersehbar ist. JackF verzichtet auf diese Eigenschaften und verwendet stattdessen zur Kommunikation Events, welche von den einzelnen Threads an die Services geschickt werden. Diese werden im Anschluss vom `MainThread` abgearbeitet. Das Versenden eines Events wird im Sequenzdiagrammen durch Run() ersichtlich gemacht. Zusätzlich dazu wird der `MainThread` durch ▨ und ein Workerthread durch ▩ gekennzeichnet. Da JackF mehrere Analysen für mehrere Anzeigemöglichkeiten „gleichzeitig“ durchführt, ist es notwendig, einem Thread den aktuellen Status jeder Analyse zur Verfügung zu stellen. Dazu wird für jede Anfrage ein `nsIJackFControlManagerObject` Objekt angelegt. Dieses Objekt ist eine Art Arbeitspaket, welches alle für die Abarbeitung einer Analyse relevanten Daten und Verweise beinhaltet.

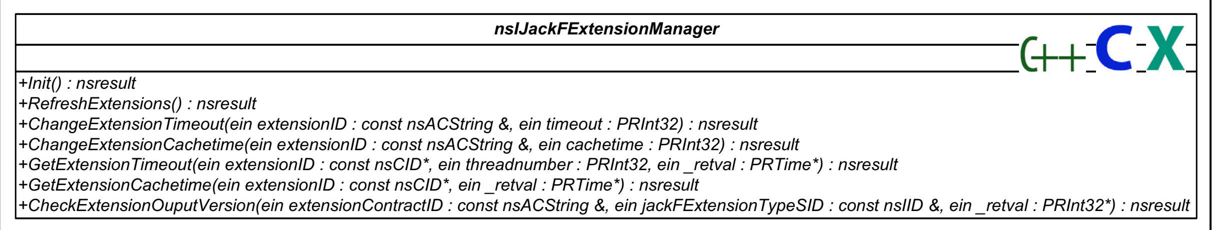

Abbildung 4.10: Interface mit Klasse und XPCOM in C++

4.4.1 Display Klassen

Unter dem Sammelbegriff Display Klassen werden alle Klassen zusammengefasst, die für die Darstellung von Analyseergebnissen verantwortlich sind. Das gewählte Design ist ein klassisches Beispiel für die strikte Trennung der Präsentationsschicht von der Anwendungslogik [KrPo88]. Dies erlaubt es, später problemlos weitere Darstellungsmöglichkeiten hinzuzufügen. Darstellen beschränkt sich dabei nicht auf die bildhafte Ausgabe sondern könnte, auch wenn in diese Richtung noch keine Versuche unternommen wurden, Personen mit Sehbehinderung auf besonders gefährliche Nachrichten, zum Beispiel mittels Ausgabe eines Warnsignals über einen Lautsprecher, hinweisen.

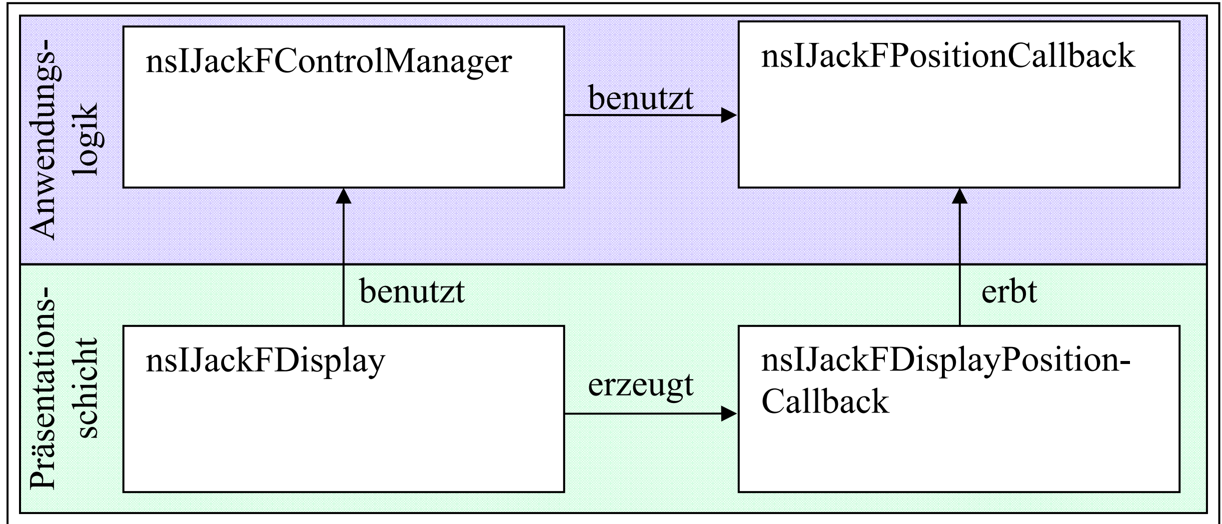

Abbildung 4.11: Trennung von Präsentationsschicht und Anwendungslogik

Die Abbildung zeigt das Design der Display Klassen auf höchster Abstraktionsebene. Dadurch soll veranschaulicht werden, wie zusätzliche Ausgabemöglichkeiten hinzugefügt werden können, ohne die Anwendungslogik darauf anzupassen. Die in der Abbildung `nsIJackFDisplay` genannte Komponente stellt in den meisten Fällen ein Overlay dar, welches auf die Benutzeroberfläche von Thunderbird gelegt wird und das Anzeigen von Analyseergebnissen übernimmt. Die anzuzeigenden Analyseergebnisse werden vom `nsIJackF-`

`ControlManager` erfragt, welcher die eigentliche Anwendungslogik beinhaltet. Ist das Analyseergebnis nicht bereits in der Datenbank gespeichert und muss somit berechnet werden, wird dies `nsIJackFDisplay` mitgeteilt. `nsIJackFControlManager` hat im Anschluss die Möglichkeit, durch ein Objekt einer Klasse, welche das Interface `nsIJackFPositionCallback` implementiert, `nsIJackFDisplay` darüber zu informieren, dass ein Analyseergebnis der zuvor gestellten Anfrage vorliegt. Im obigen Beispiel wäre dieses Objekt von der Klasse `nsIJackFDisplayPositionCallback`, welche die Aufgabe hat, das Display zum erneuten Anfragen des Analyseergebnisses aufzufordern, um die Benutzeroberfläche zu aktualisieren.

Das Design der Display Klassen wird anhand der für die Darstellung der Analyseergebnisse in einer Spalte in der Nachrichtenübersicht zuständigen Klassen, stellvertretend für alle anderen Möglichkeiten wie Lesebereich und Mouse-Over-Events, beschrieben.

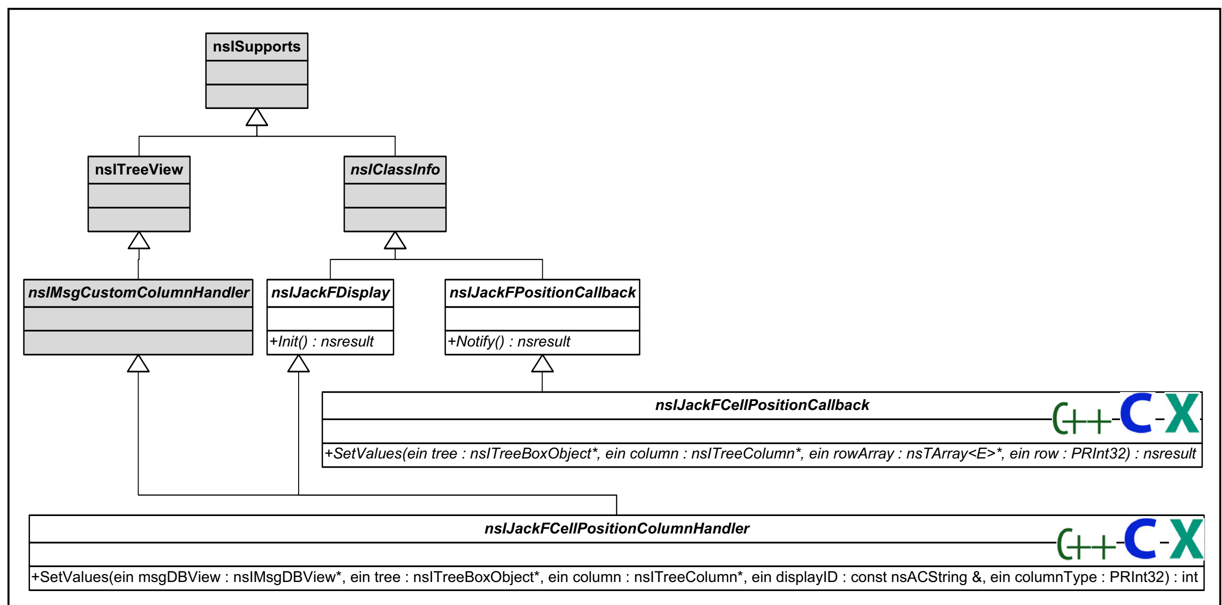

Abbildung 4.12: Klassendiagramm - Display Klassen

Das unten abgebildete Sequenzdiagramm zeigt die einfachste Form einer Anfrage, bei welcher das Analyseergebnis bereits in der Datenbank gespeichert ist. Dieser Teil von JackF wird jedes Mal ausgeführt, wenn eine Anfrage um ein Analyseergebnis gestellt wird. Dies kann durch ständiges Aktualisieren der anzuzeigenden Informationen sehr häufig erfolgen. Aus diesem Grund ist es notwendig, diesen Vorgang besonders performant zu programmieren. In Kapitel *4.5 Designentscheidungen und ausgewählte Codefragmente* werden einige Beispiele dafür gegeben, wodurch dieser Vorgang beschleunigt wurde. Wie in diesem Beispiel zu sehen ist, wird ein `nsIJackFCellPositionCallback` Objekt angelegt. Das wäre in diesem Fall

nicht zwingend notwendig gewesen, denn das Callback wird nur benötigt, wenn für eine Analyse noch kein Analyseergebnis in der Datenbank gespeichert ist. Da jedoch zum letztmöglichen Zeitpunkt, zu dem das Anlegen geschehen kann, noch nicht feststeht, ob ein Analyseergebnis gespeichert ist, muss das Objekt zwingend erzeugt werden.

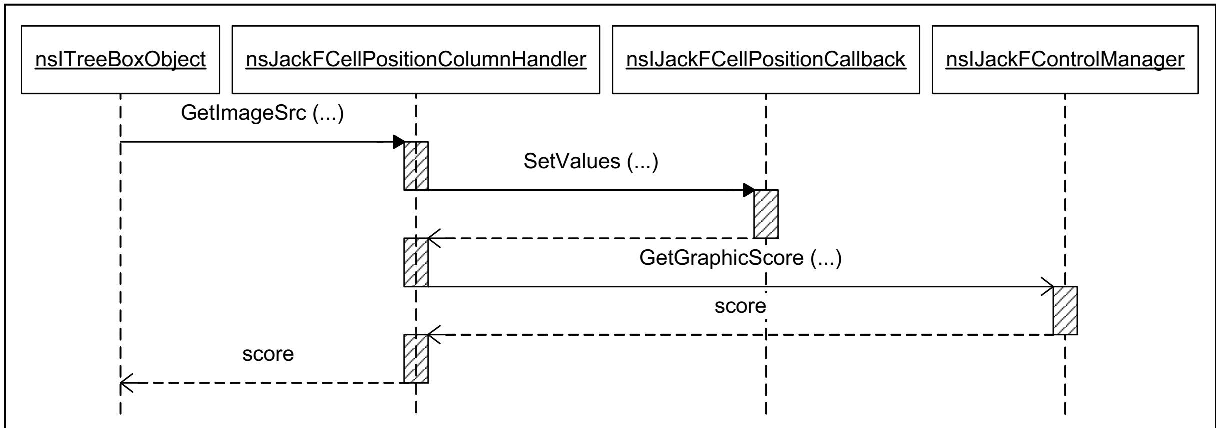

Abbildung 4.13: Sequenzdiagramm - Einfache Anfrage mit zwischengespeichertem Analyseergebnis

Für jede noch nicht gespeicherte Analyse oder für den Fall, dass das gespeicherte Analyseergebnis bereits seine Gültigkeit verloren hat, wird das Abarbeiten von Analyseergebnisabfragen aufwendiger. Zu Beginn ist der Ablauf etwa gleich, der einzige Unterschied besteht darin, dass anstatt des Analyseergebnisses von `GetGraphicScore(…)` der Hinweis zurückgegeben wird, dass das Analyseergebnis noch nicht vorliegt. Wie darauf reagiert wird, kann von der jeweiligen Display Klasse entschieden werden. Bei der Ausgabe in einer Spalte eignet sich die Anzeige eines „?“ für numerische und alphanumerische oder ein für grafische Darstellung. Hier ist wichtig zu erkennen, dass die Benutzeroberfläche nach dem Beenden von `GetGraphicScore(…)` wieder auf Benutzereingaben reagieren kann. Erst nach der im Hintergrund ablaufenden Analyse der Nachricht und dem erfolgreichen Abspeichern des Analyseergebnisses in der Datenbank wird von `nsIJackFControlManager` das `Notify()` von `nsIJackFCellPositionCallback` aufgerufen. Das Objekt von `nsIJackFCellPositionCallback` beinhaltet Informationen darüber, welche Zelle fertig analysiert wurde, und benachrichtigt die Benutzeroberfläche. Der restliche Ablauf ist wieder mit dem Anfragen eines gespeicherten Analyseergebnisses identisch.

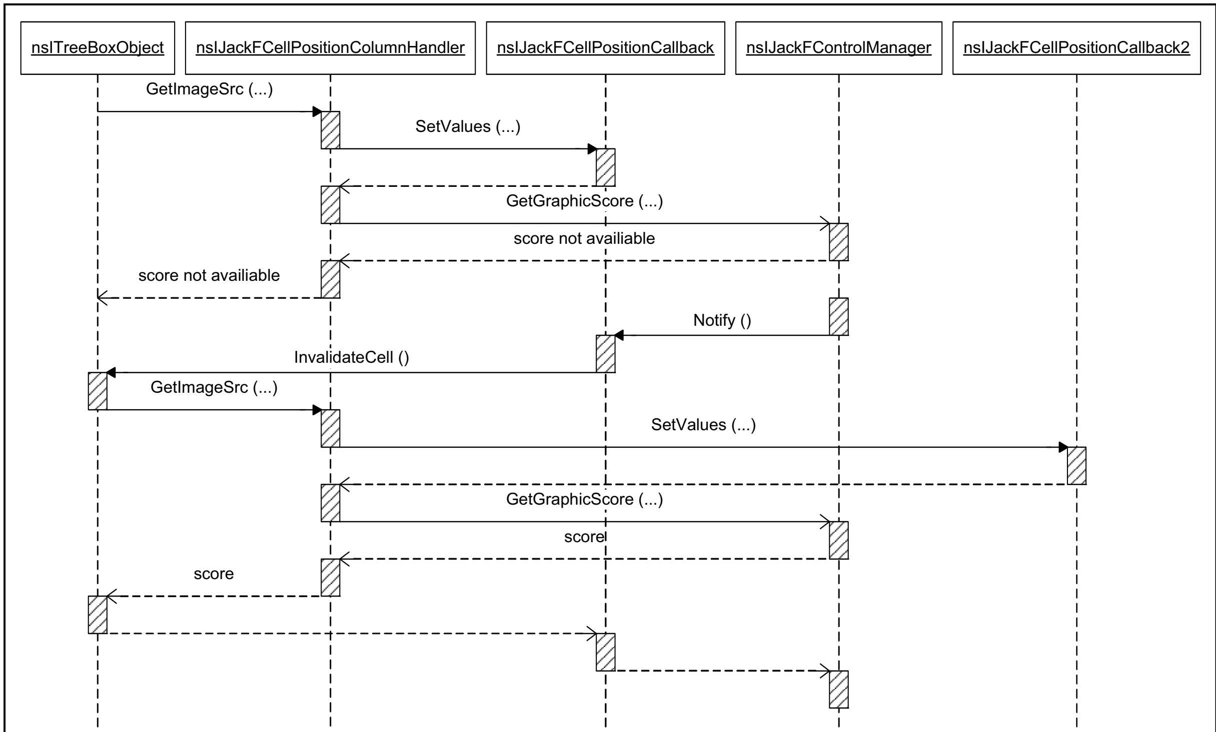

Abbildung 4.14: Sequenzdiagramm - Anfrage ohne zwischengespeichertem Analyseergebnis

Das Interface `nsIJackFCellPositionCallback` wird von `nsIJackFPositionCallback` abgeleitet, womit sichergestellt ist, dass dem `nsIJackFControlManager` immer eine `Notify()` Funktion zur Verfügung steht, ungeachtet dessen, ob die Ausgabe nun in einer Spalte in der Nachrichtenübersicht oder an einem anderen Ort erfolgt. Wie `Notify()` tatsächlich realisiert ist und welche Attribute in der jeweiligen Ableitung von `nsIJackFPositionCallback` gespeichert sind, ist für einen Aufruf ohne Parameter unerheblich.

4.4.2 Datenbank Klassen

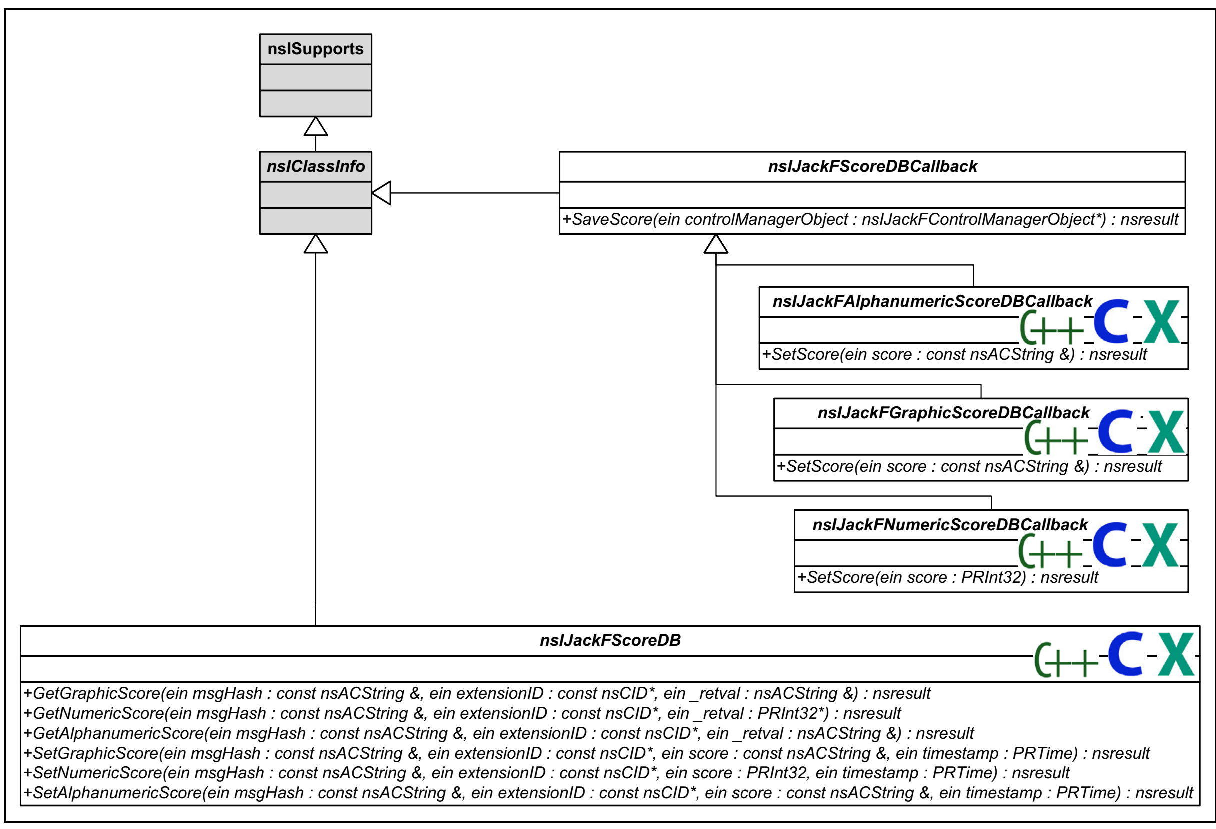

Abbildung 4.15: Klassendiagramm - Datenbank Klassen

Die Klasse `nsIJackFScoreDB` ist zur Gänze in C++ implementiert. Die Funktionen `GetGraphicScore(…)`, `GetNumericScore(…)` und `GetAlphanumericScore(…)` werden dazu verwendet, gespeicherte Analyseergebnisse aus der Datenbank zu laden. Dabei ist zu beachten, dass nur jene Analyseergebnisse geladen werden, die zum Zeitpunkt der Anfrage noch gültig sind.

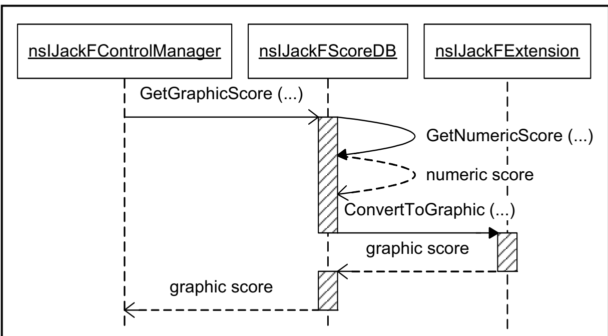

Abbildung 4.16: Sequenzdiagramm - Umrechnung eines numerischen Analyseergebnisses in ein grafisches

Sollte ein grafisches oder alphanumerisches Analyseergebnis abgefragt werden und dieses nicht vorhanden sein, kann versucht werden, das numerische Analyseergebnis abzufragen und

dieses im Anschluss mit den Funktionen ConvertToGraphic(…) bzw. ConvertToAlphanumeric(…) des geforderten Pluggys umzurechnen. Nicht alle Pluggies unterstützen die Umrechnung eines numerischen Analyseergebnisses in ein anderes. Es ist Entwicklern von Pluggies, die eine längere Laufzeit oder hohen Ressourcenbedarf aufweisen, anzuraten, von dieser Möglichkeit Gebrauch zu machen, um das System zu entlasten. Damit nicht für jede Umrechnung ein neues Objekt eines Pluggys angelegt werden muss, ist es notwendig, bereits beim Starten des Datenbankservices, von allen Pluggies, die eine Umrechnung unterstützen, ein Objekt anzulegen und dieses bei Bedarf wieder zu verwenden.

Grundlegend wäre es nsIJackFControlManager durch Aufrufen einer SetScore(…) Funktion möglich, Werte in die Datenbank zu speichern. Dies würde allerdings die Implementierung von nsIJackFControlManager wesentlich komplexer gestalten, da dieser dafür zwischen numerischen, alphanumerischen und grafischen Analyseergebnissen unterscheiden und in Abhängigkeit des gegebenen Typs die richtige Funktion aufrufen müsste. Aus diesem Grund werden zum Speichern nsIJackFScoreDBCallback Objekte, ähnlich der Objekte der Display Klassen, verwendet. Nach dem Starten einer Analyse ist dem Workerthread ohnehin bekannt, ob es sich um ein grafisches, numerisches oder alphanumerisches Analyseergebnis handelt. Dazu passend wird entweder ein nsIJackFGraphicScoreDBCallback, nsIJackFNumericScoreDBCallback oder nsIJackFAlphanumericScoreDBCallback Objekt an- und das Analyseergebnis darin abgelegt. Bevor nun nsIJackFControlManager über das Vorliegen eines Analyseergebnisses in Kenntnis gesetzt wird, wird im nsIJackFControlManagerObject Objekt ein Verweis auf das angelegte nsIJackFScoreDBCallback Objekt abgespeichert. Der nsIJackFControlManager kann nun durch Aufrufen der Funktion Save() des im nsIJackFControlManagerObject Objekt gespeicherten nsIJackFScoreDBCallback Objektes das Analyseergebnis in die Datenbank speichern.

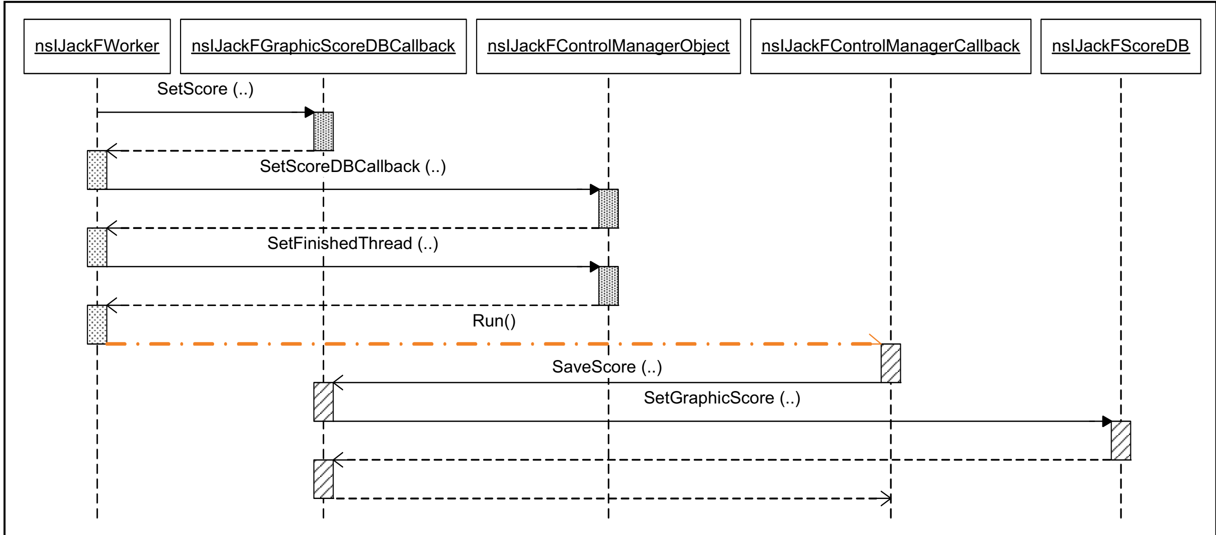

Abbildung 4.17: Sequenzdiagramm - Speichern eines Analyseergebnisses

Da dieser Aufruf von `Save()` ausschließlich vom `MainThread` erfolgt, muss die Datenbank nicht als Threadsafe gekennzeichnet oder „reentrant" implementiert werden. Bei der in der aktuellen Version von JackF befindlichen Datenbank handelt es sich nicht mehr um eine RDF Datei sondern um die in Thunderbird integrierte frei erhältliche SQLite Datenbank.

4.4.3 Konfigurationsklassen

Die Hauptaufgabe des `nsIJackFExtensionManager` ist es, alle Pluggies zu verwalten. Dies beinhaltet das Registrieren neuer sowie das Entfernen nicht mehr installierter Pluggies. Zusätzlich wird für jedes Pluggy die gewünschte Zeitspanne, in der ein zwischengespeichertes Analyseergebnis gültig ist und dessen Time-out, in dem eine Analyse beendet sein muss, gespeichert. `nsIJackFExtensionsRDFManager` ist in JS implementiert, da dieser von JackF Version 1.0 übernommen wurde. Eine Portierung der Component nach C++ ist nicht notwendig, da dadurch keine Effizienzsteigerung möglich wären.

Auch wenn sich die Funktionen von `nsIJackFExtensionManager` und `nsIJackFExtensionsRDFManager` sehr ähneln, ist eine Trennung der beiden sinnvoll. Wie bereits bei der Weiterentwicklung von JackF Version 1.0 zu Version 2.0 erkannt wurde, ist es dadurch möglich, nur häufig verwendete Teile von JackF durch in C++ optimierte Components zu ersetzen und die weniger häufig verwendeten im langsameren, aber einfacher zu wartenden JS zu belassen.

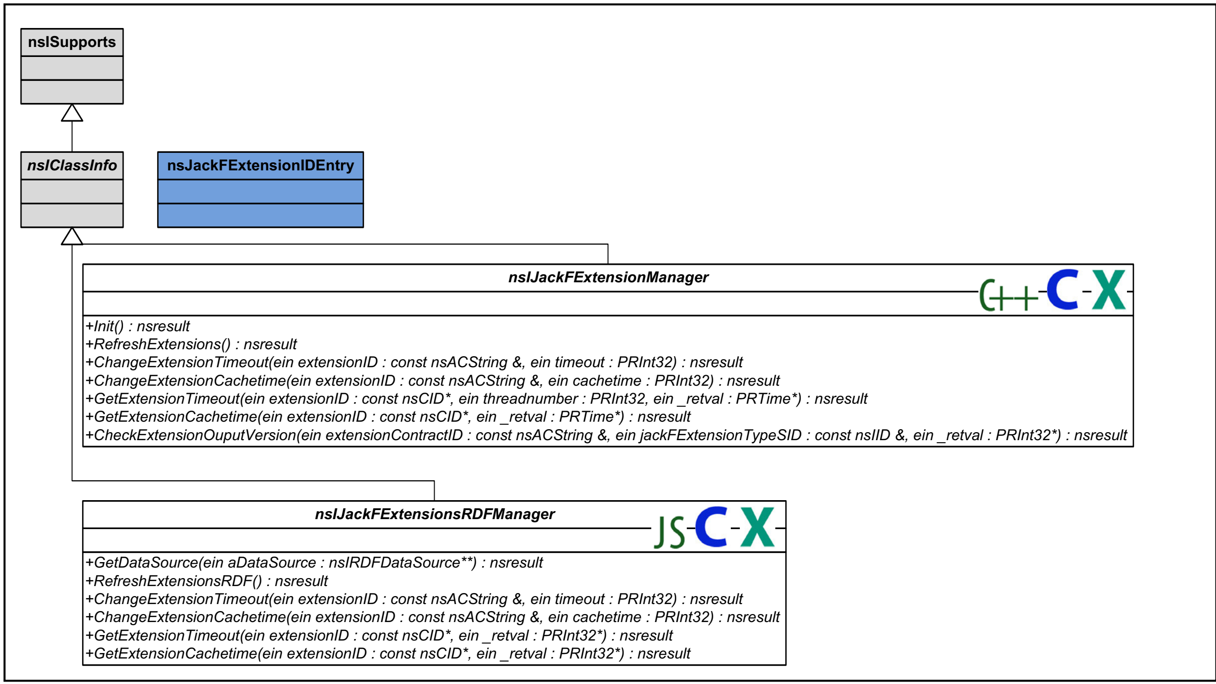

Abbildung 4.18: Klassendiagramm - Konfigurationsklassen Extensionmanager

JackF speichert die Konfiguration der Pluggies in einer RDF Datei ab. Dies ist notwendig, da ausschließlich auf diese Weise mit XUL auf die Daten zugegriffen werden kann, und diese in der Benutzeroberfläche dargestellt werden können. Die Manipulation von RDF Dateien wäre in C++ im Vergleich zu JS relativ aufwendig. Mit JS kann dieser Aufwand in Grenzen gehalten werden, dennoch ist der Ressourcenbedarf für das ständige Laden von Werten aus einer RDF Datei beträchtlich.

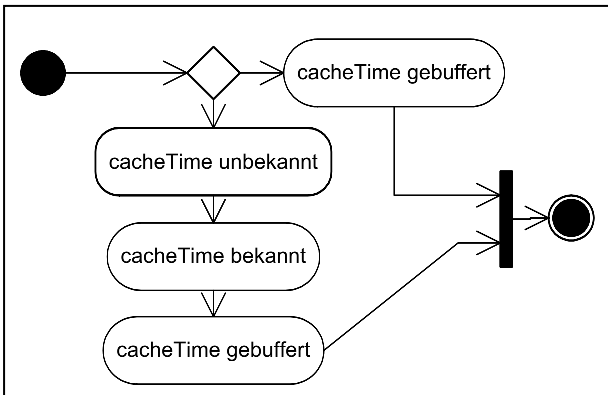

Abbildung 4.19: Zwischenspeicher zur Performancesteigerung von RDF Datenhaltung

Aus diesem Grund wurde im `nsIJackFExtensionManager`, welcher in C++ implementiert wurde, ein Zwischenspeicher eingebaut, der bereits einmal benötigte Informationen, die aus der RDF Datei geladen wurden, zwischenspeichert. Dies macht ein wiederholtes Lesen der RDF Datenbank unnötig. Der Zwischenspeicher ist in Form einer Liste von `nsExtensionI-DEntry` Objekten aufgebaut. Bei der Klasse `nsExtensionIDEntry` handelt es sich um kei-

ne Component sondern um eine ausschließlich von `nsIJackFExtensionManager` verwendbare C++ Klasse.

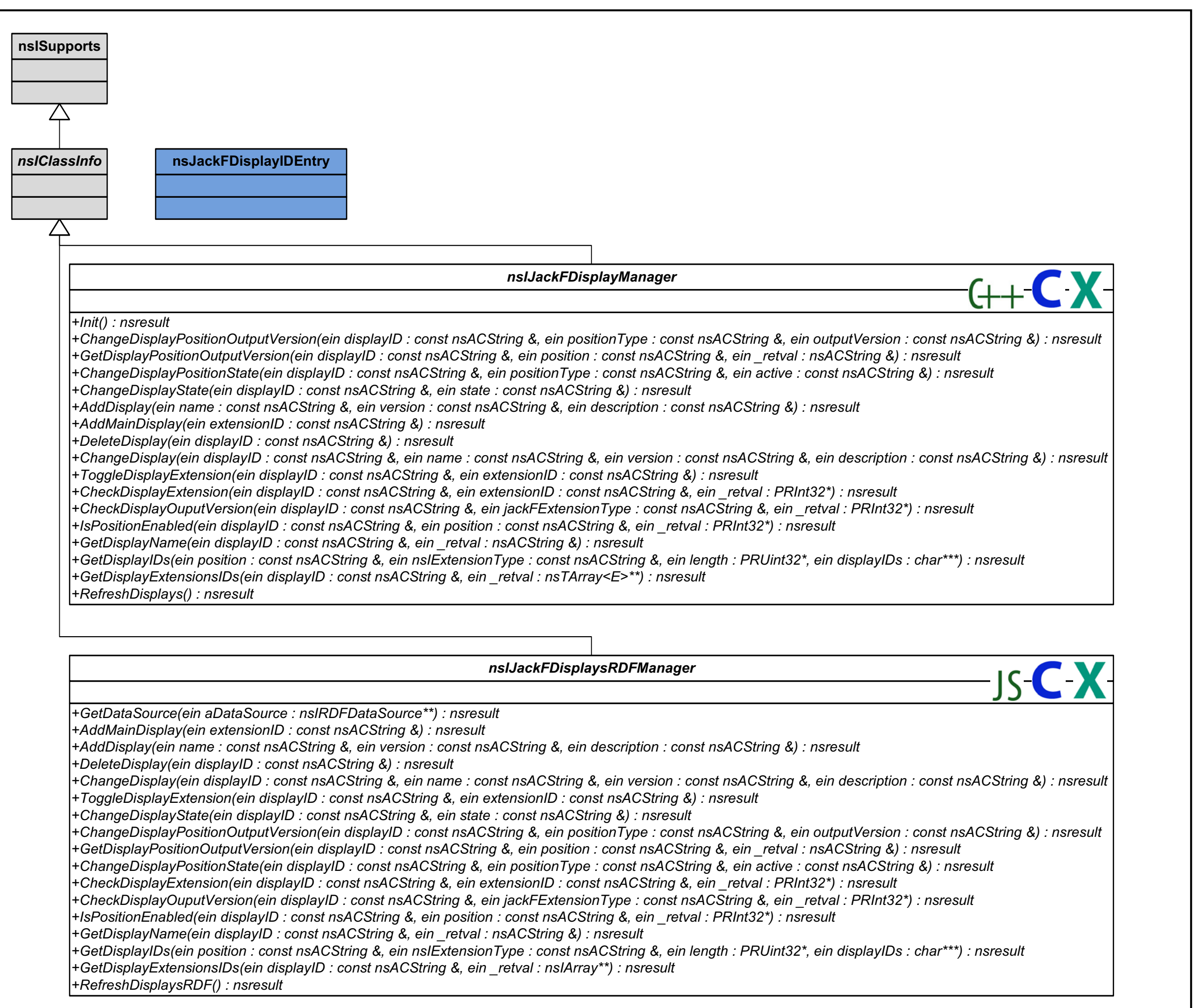

Abbildung 4.20: Klassendiagramm - Konfigurationsklassen Displaymanager

Der Aufbau des `nsIJackFDisplayManager` ähnelt dem des `nsIJackFExtensionManager`. Auch hier wurde basierend auf derselben Überlegung die Manipulation der RDF Datei in einer JS Component belassen. Die Component ist aufwendiger als der `nsIJackFExtensionManager`, da Funktionen für die Benutzeroberfläche bereitgestellt werden müssen. So ist etwa `changeDisplayState(…)` für das Ein- bzw. Ausschalten eines Displays verantwortlich. Die Gestaltung der Benutzeroberfläche wird dadurch wesentlich vereinfacht, da beim Klicken auf einen Button lediglich eine Funktion aufgerufen werden muss.

4.4.4 JackF Kernstück

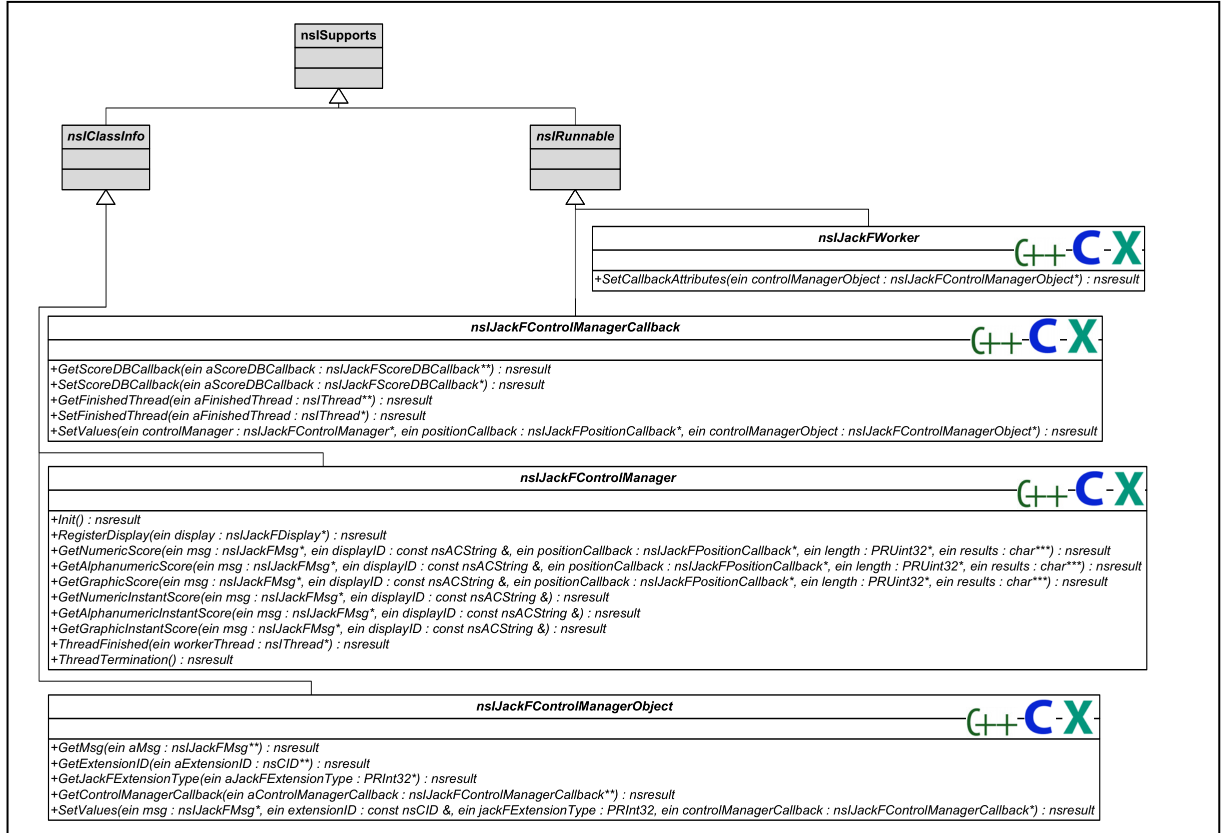

Abbildung 4.21: Klassendiagramm – JackF Kernstück

Die Component `nsIJackFControlManager` ist das eigentliche Kernstück von JackF. Es übernimmt und steuert den Großteil der Aufgaben, die für eine erfolgreiche Analyse durchzuführen sind. Diese umfassen das Anfragen von Analyseergebnissen aus der Datenbank, das Starten und Beenden von Threads, die Verwaltung der abzuarbeitenden Analysen in einer Warteschlange sowie das Beenden zu lange arbeitender Threads.

Wie bereits in Kapitel *4.4.1 Display Klassen* beschrieben wurde, tritt folgendes Szenario am häufigsten auf: eine Display Klasse stellt eine Anfrage an JackF, wobei das Analyseergebnis bereits in der Datenbank zwischengespeichert ist. Sollte kein Analyseergebnis zwischengespeichert sein, muss JackF dies der Display Klasse mitteilen und eine Analyse durch einen im Hintergrund arbeitenden Thread veranlassen. Wie das unten abgebildete Sequenzdiagramm zeigt, wird vor dem Erzeugen eines Workerthreads von `nsIJackFControlManager` geprüft, ob bereits die höchstzulässige Anzahl an Threads genutzt wird. Sind zusätzliche Threads verfügbar, wird ein solcher angelegt und ein Objekt der Klasse `nsIJackFWorker` erzeugt, welches die Analyse veranlasst. Damit die Benutzeroberfläche während der Analyse auf Benutzereingaben reagieren kann, teilt `nsIJackFControlManager` der Display Klasse mit, dass kein Analyseergebnis vorliegt, und eine Analyse veranlasst wurde. Sobald dem

Workerthread ein Analyseergebnis vorliegt, benutzt dieser nsIJackFControlManager-Callback, um eine Bearbeitung des Analyseergebnisses im MainThread zu veranlassen. Diese Vorgehensweise ist notwendig, da sowohl die Datenbank als auch alle Elemente der Benutzeroberfläche ausschließlich vom MainThread aus benutzt werden dürfen. Nachdem das Analyseergebnis gespeichert wurde, und über nsIJackFPositionCallback die Display Klasse über das Vorliegen eines Analyseergebnisses benachrichtigt wurde, wird von nsIJackFControlManager geprüft, ob weitere unbeantwortete Anfragen existieren. Ist eine Anfrage in der Warteschlange, wird der bereits existierende Thread benutzt, um eine weitere Analyse durchzuführen. Ist wie im unten abgebildeten Szenario keine Anfrage in der Warteschlage, wird der Workerthread beendet und erst bei Bedarf neu erzeugt. JackF kann an zukünftige Prozessorgenerationen mit Mehrkerntechnologie anpasst werden. Dazu ist es möglich, die höchstzulässige Anzahl an Threads frei zu wählen.

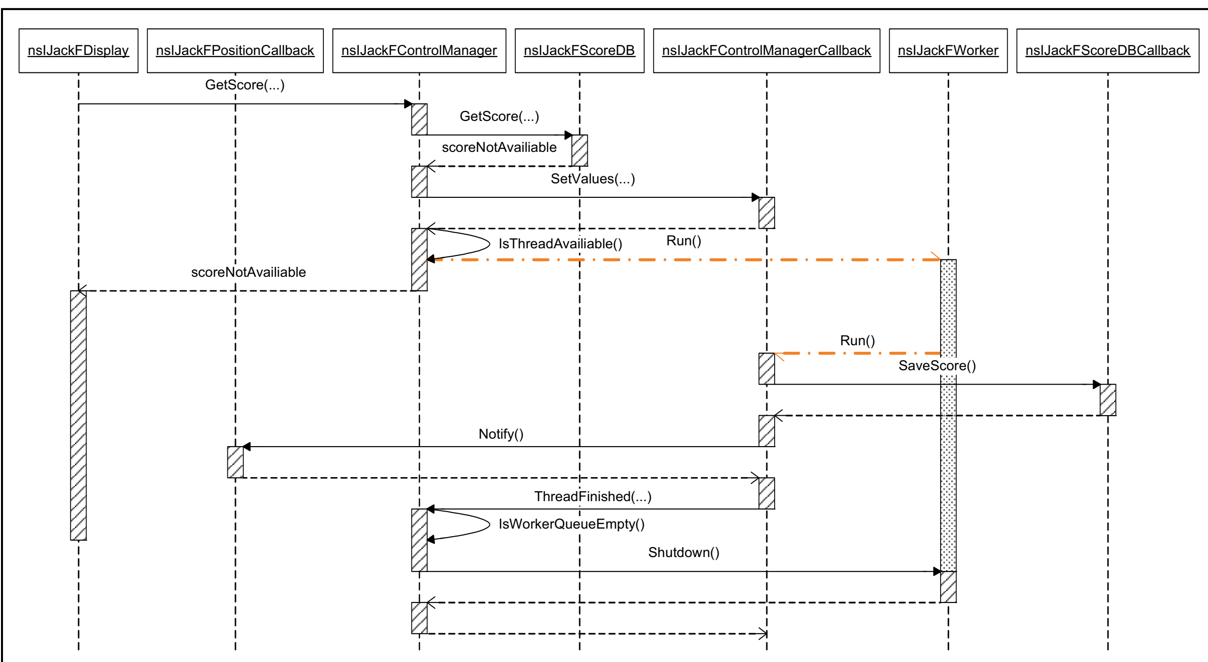

Abbildung 4.22: Sequenzdiagramm – JackF Kernstück

nsIJackFControlManager geht davon aus, dass bei hohem Anfrageaufkommen die zuletzt gestellten Anfragen für den Anwender von höherer Dringlichkeit und größerem Interesse sind als die schon länger in der Warteschlange befindlichen. Dies wäre zum Bespiel der Fall, wenn über die Nachrichtenübersicht geblättert wird, und unbeantwortete Anfragen bereits aus dem sichtbaren Bereich verschwinden, bevor sie analysiert wurden. Durch das Organisieren der Warteschlange nach dem last-in-first-out Prinzip wird versucht, immer die dringlichsten Anfragen zuerst abzuarbeiten. nsIJackFControlManager besitzt eine Warteschlange für alle

Display Klassen und unterscheidet nicht, von welcher Display Klasse eine Anfrage gestellt wurde. Es werden alle Display Klassen gleich behandelt.

4.4.5 Worker Klassen

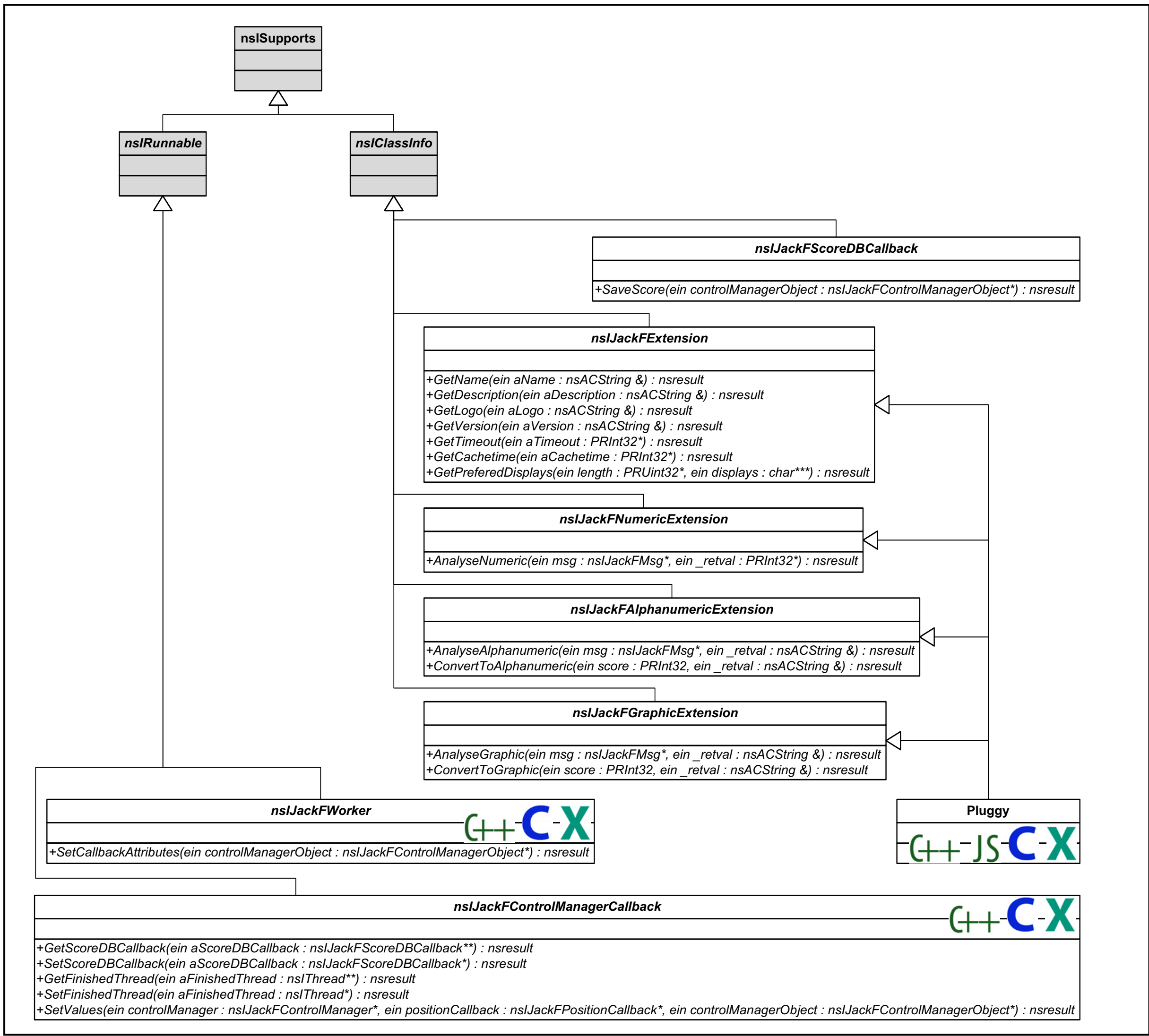

Abbildung 4.23: Klassendiagramm – Worker Klassen

Im folgenden Sequenzdiagramm wird der Ablauf vom Starten eines Threads bis zum Senden der Nachricht, dass die Analyse beendet wurde, an den `MainThread` beschrieben. Nachdem `nsIJackFWorker` seine Arbeit aufgenommen hat, wird ein Objekt des Pluggys erzeugt, welches für die Analyse der Anfrage verantwortlich ist. Anschließend wird vom Pluggy die benötigte Analyse durchgeführt. Je nachdem ob ein grafisches, numerisches oder alphanumerisches Analyseergebnis gefordert wird, wird ein zum Analyseergebnis passendes `nsIJackF-ScoreDBCallback` angelegt und das Analyseergebnis darin abgespeichert. `nsIJackF-ScoreDBCallback` wird, wie in Kapitel *4.4.2 Datenbank Klassen* beschrieben, von `nsI-`

JackFControlManager benutzt, um das Analyseergebnis in die Datenbank zu speichern. Da nsIJackFControlManager wissen muss, welcher Thread seine Arbeit beendet hat, muss eine Referenz auf nsIJackFWorker in nsIJackFControlManagerCallback abgelegt werden, bevor die Nachricht über das Beenden der Analyse an den MainThread gesandt wird.

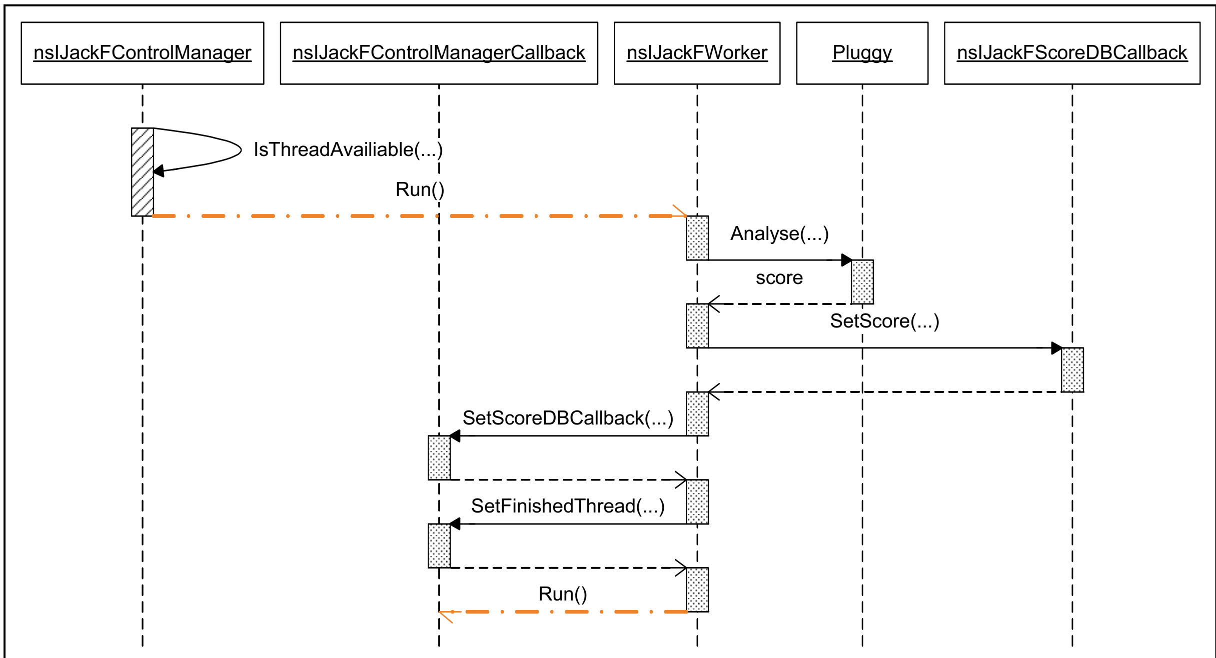

Abbildung 4.24: Sequenzdiagramm - Kommunikation Thread mit MainThread

4.4.6 Hilfsklassen

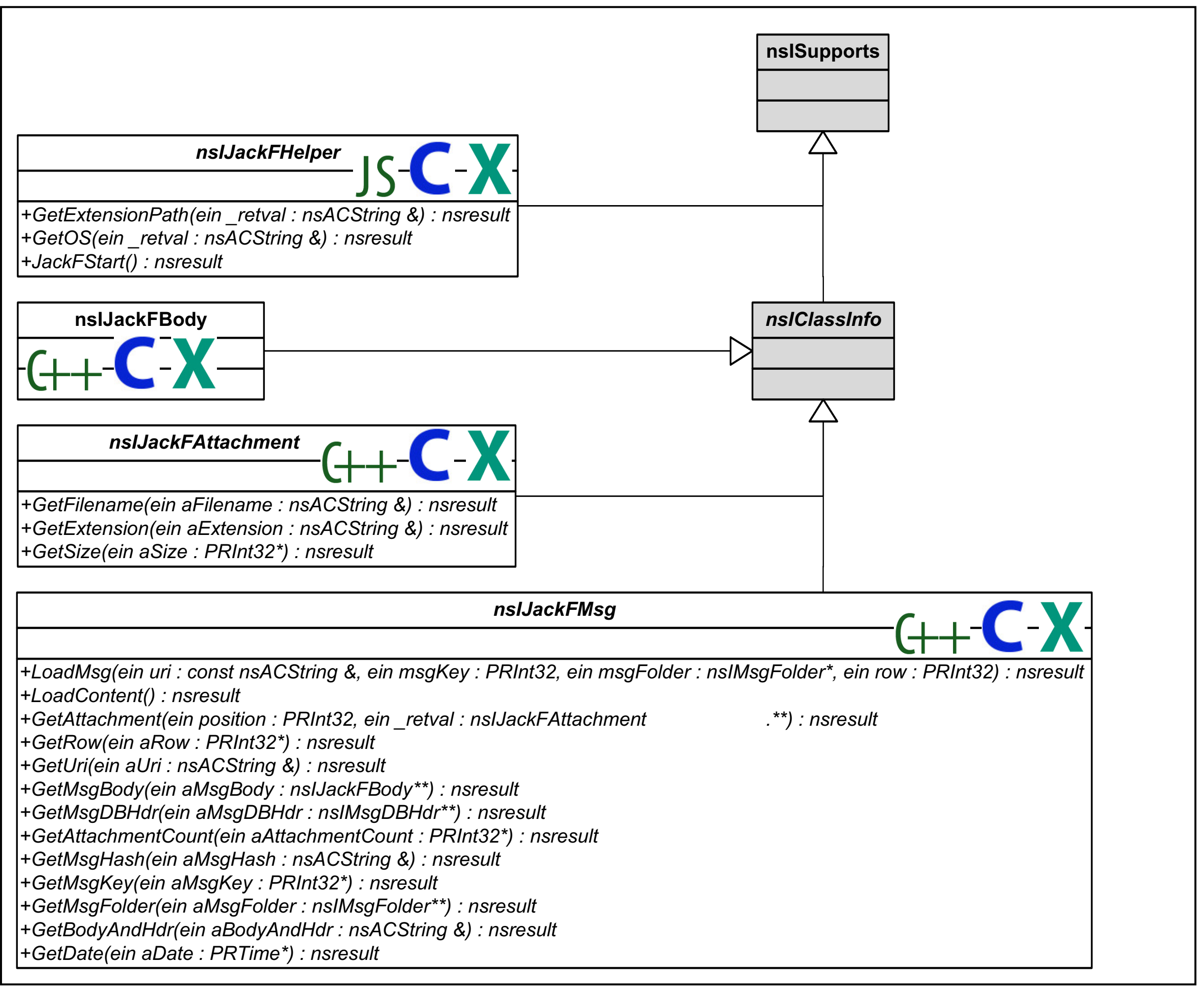

Abbildung 4.25: Klassendiagramm - Hilfsklassen

Bei den Hilfsklassen gibt es zwei Kategorien. Die eine fasst Components, die das Entwickeln von Pluggies erleichtern zusammen, die andere enthält die Component `nsIJackFHelper`, die Hilfsfunktionen für JackF bereitstellt. Letztere initialisiert beim Aufruf von `JackFStart()` alle Komponenten von JackF und bereitet das Framework auf Anfragen vor. Die Funktion `GetExtensionPath()` wird verwendet, um den Pfad des Installationsverzeichnisses von JackF zu ermitteln. Dieser wird benötigt, da der Datenspeicher der Datenbank und alle RDF Dateien in diesem abgelegt werden. Um auf verschiedenen Betriebsystemen einsetzbar zu sein, wird mit der Funktion `GetOS()` das Betriebssystem ermittelt und gegebenenfalls in Linux Betriebssystemen alle „\“ durch „/“ ersetzt.

Die zweite Kategorie besteht aus den Components `nsIJackFMsg`, `nsIJackFAttachment` und `nsIJackFBody`. Diese werden von Pluggies verwendet, um komfortabler auf eine Nachricht oder Teile dieser zugreifen zu können.

4.4.7 Zusätzliche Diagramme

4.4.7.1 JackFExtensionRegistration

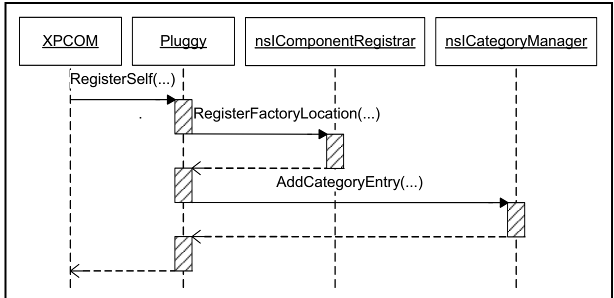

Abbildung 4.26: Sequenzdiagramm – Pluggy Registrierung

Beim Starten von Thunderbird und XPCOM wird jede einzelne Factory aller Pluggies geladen und registriert. Im Zuge dessen registriert sich jedes Pluggy zusätzlich in dem von Thunderbird zur Verfügung gestellten `nsICategorieManager` Service. Dieser Service wird von JackF benutzt, um alle installierten Pluggies zu erfragen oder nicht mehr installierte Pluggies aus JackFs Konfiguration zu entfernen.

4.4.7.2 Analyseabfrage auf höchstem Detaillierungsgrad

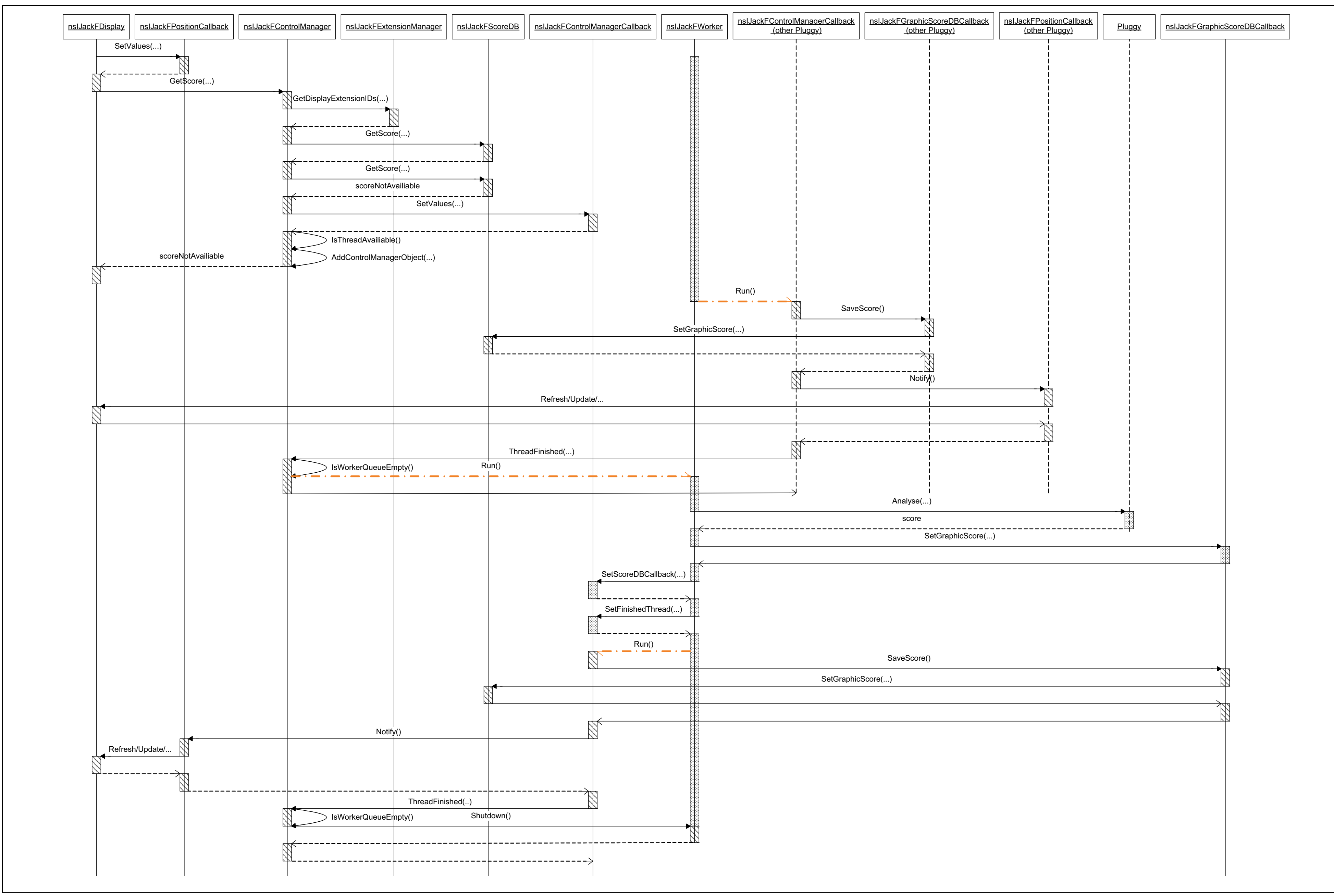

Abbildung 4.27: Sequenzdiagramm – Analyseabfrage auf höchstem Detaillierungsgrad

Die obige Abbildung zeigt das Abfragen eines Analyseergebnisses auf höchstem Detaillierungsgrad. Wie ersichtlich ist, wird im Sequenzdiagramm davon ausgegangen, dass weder ein Analyseergebnis zwischengespeichert ist, noch dass ein freier Thread zum Abarbeiten der Analyse zur Verfügung steht. Sobald ein `nsIJackFWorker` seine Analyse beendet hat, und das Display durch `nsIJackFPositionCallback` über das Vorliegen eines Analyseergebnisses in Kenntnis gesetzt wurde, wird `nsIJackFWorker` von `nsIJackFControlManager` weiterverwendet und die Analyse der in der Warteschlange befindlichen Anfrage veranlasst.

4.5 Designentscheidungen und ausgewählte Codefragmente

In diesem Kapitel möchte der Autor auf einige besonders effizienzsteigernde Maßnahmen bei der Realisierung von JackF genauer eingehen. Zusätzlich wird auf Designentscheidungen, welche aus den Klassen- und Sequenzdiagrammen nicht ersichtlich sind, näher eingegangen.

4.5.1 Analyseergebnisse in einer Spalte in der Nachrichtenübersicht

Die Ausgabemöglichkeit von Analyseergebnissen in einer Spalte in der Nachrichtenübersicht war bei der Implementierung von JackF eine besondere Herausforderung. Zwar ist die Komplexität der für diese Ausgabemöglichkeit benötigten Components klein, jedoch kann durch die sehr unangenehme Eigenschaft von Thunderbird, Zellen in der Nachrichtenübersicht unnötig häufig zu aktualisieren, ein Performanceproblem entstehen.

Das Initialisieren dieser Ausgabemöglichkeit wird durch ein Overlay realisiert. Beim Starten von Thunderbird, wenn das Overlay auf der gewünschten Stelle platziert wird, werden einige im Overlay gespeicherte JS Anweisungen ausgeführt, welche die Initialisierung vornehmen. Bei diesem Vorgang wird bei `nsIJackFControlManager` angefragt, wie viele Spalten zur Darstellung von Analyseergebnissen benötigt werden. Diese werden in die Nachrichtenübersicht eingefügt. Die Möglichkeit eine eigene Spalte in Thunderbird einzufügen, wurde mit der Version 3.0pre1 stark vereinfacht. Dazu wurde das Interface `nsIMsgCustomColumnHandler` eingeführt, welches alle für die Steuerung einer Spalte benötigten Funktionen beinhaltet. Um einer selbst hinzugefügten Spalte Funktionalität zu geben, muss eine Component, welche `nsIMsgCustomColumnHandler` implementiert, entwickelt werden. Dies wird bei JackF mit `nsIJackFCellPositionColumnHandler` realisiert. Ursprünglich wurde während der Initialisierung für jede eingefügte Spalte umgehend ein `nsIJackFCellPositionColumn-`

Handler Objekt erzeugt und der Spalte zugeordnet. Dies erwies sich zu einem späteren Zeitpunkt als untauglich, da es notwendig ist, beim Wechsel von einem Nachrichtenordner in einen anderen die Spalten neu zu initialisieren. Die Initialisierung erfolgt daher nun in zwei Schritten. Zu Beginn werden alle benötigten Spalten eingefügt und in einer Liste eine Referenz auf diese gespeichert.

Für den zweiten Teil der Initialisierung ist es notwendig zu wissen, dass in Thunderbird die Möglichkeit besteht, einen sogenannten Observer für das Auftreten bestimmter Ereignisse zu registrieren. Dieser Observer wird beim Eintreten des gewählten Ereignisses benachrichtigt und kann auf dieses reagieren. Für diesen Zweck wird bei der Initialisierung ein JS Objekt angelegt, welches als Observer für das Ereignis mit dem Namen MsgCreateDBView fungiert. Dieses Ereignis tritt immer ein, wenn in einen anderen Nachrichtenordner gewechselt wird. Der Observer nutzt die in Schritt eins in der Liste abgespeicherten Referenzen, erzeugt für jede Spalte ein neues Objekt von nsIJackFCellPositionColumnHandler und weist es einer Spalte zu. Durch diese Vorgehensweise ist es beim Wechseln auf einen neuen Nachrichtenordner möglich, eine Spalte neu zu initialisieren.

Es wurde versucht, alle Components dieser Ausgabemöglichkeit in C++ zu implementieren. Dies konnte allerdings nicht realisiert werden, da in C++ nicht auf die in JS globale Variable mit dem Namen gDBView zugegriffen werden konnte. gDBView wird benötigt, um eine Nachricht aus der Datenbank von Thunderbird zu laden und den Pluggies zur Verfügung zu stellen. Auch wenn es in C++ keinen Verweis auf gDBView gibt, kann dieser im zweiten Schritt der Initialisierung, von JS an eine C++ Component übergeben werden. Dieses Vorgehen ermöglicht es, die häufig benutzten Teile der Ausgabemöglichkeit in C++ zu entwickeln und eine Performancesteigerung zu erreichen.

Der Inhalt einer Zelle wird von Thunderbird sehr häufig aktualisiert. Dieser Umstand ist, wenn in einer Zelle ein bereits zwischengespeichertes Analyseergebnis dargestellt werden soll, nicht bedenklich. Soll in der Zelle jedoch ein noch nicht in der Datenbank gespeichertes Analyseergebnis ausgegeben werden, ist ohne geeignete Vorkehrungen folgendes Szenario vorstellbar: Der Anwender scrollt in der Nachrichtenübersicht zu einer Nachricht, deren Analyseergebnisse noch nicht in der Datenbank gespeichert sind. Während die erste Anfrage in der Warteschlange gespeichert ist und auf ihre Bearbeitung wartet, bewegt der Anwender die Maus über die Zeile, in der die Nachricht angezeigt wird. Das Resultat ist, dass während die

Maus über die Zeile der Nachricht bewegt wird, zahlreiche zusätzliche Anfragen dieselbe Nachricht betreffend an `nsIJackFControlManager` gestellt und in der Warteschlage abgespeichert werden. Dies wäre eine Vergeudung von Ressourcen und würde die Benutzerfreundlichkeit stark beeinflussen.

Das Problem wurde in `nsIJackFCellPositionColumnHandler` damit gelöst, dass in einer Liste alle noch nicht beantworteten Anfragen gespeichert werden. Wird nun von Thunderbird eine Zelle, deren Analyseergebnis noch in Bearbeitung ist, erneut angefordert, wird diese von `nsIJackFCellPositionColumnHandler` nicht an `nsIJackFControlManager` weitergeleitet sondern selbst mit dem Hinweis, dass die Anfrage in Bearbeitung ist, beantwortet. Bei Versuchen mit 5.169 Nachrichten und 72.000 Anfragen stellte sich heraus, dass es zu aufwendig ist, die gesamte Liste mit schlimmstenfalls 5.169 Einträgen sequenziell zu durchsuchen um festzustellen, ob eine Anfrage bereits in der Warteschlange abgelegt ist. Die Liste ist nicht sortiert und es werden viele Objekte hinzugefügt bzw. entfernt. Aus diesem Grund hat sich der Autor entschieden, die Liste auf 255 getrennte Listen aufzuteilen, und damit den Suchaufwand im Durchschnitt auf 1/255 zu reduzieren. Der für die Verwaltung benötigte Mehraufwand ist im Vergleich zur Einsparung vernachlässigbar. Um zu beurteilen in welcher Liste eine Anfrage zu speichern oder zu suchen ist, wird die Zeilennummer der Nachricht in der Nachrichtenübersicht herangezogen und modulo 255 gerechnet.

Wie bereits in Kapitel *4.4.1 Display Klassen* beschrieben wird der Display Klasse vom `nsIJackFControlManager` mittels Aufruf der Funktion `Notify()` des `nsIJackFPositionCallback` Interfaces mitgeteilt, dass ein Analyseergebnis für eine Anfrage vorliegt. In der folgenden Abbildung wird die Realisierung von `Notify()` in `nsIJackFCellPositionCallback` gezeigt. Bei der Ausgabe eines Analyseergebnisses in einer Spalte in der Nachrichtenübersicht ist unter Umständen keine Aktualisierung des angezeigten Analyseergebnisses notwendig, da die zu aktualisierende Zelle bereits aus dem für den Anwender sichtbaren Bereich verschwunden ist. Aus diesem Grund wird der aktuell sichtbare Bereich bestimmt und, nur wenn notwendig, eine Aktualisierung vorgenommen. Es ist `Notify()` nicht möglich, das Analyseergebnis direkt in die Zelle zu schreiben. Aus diesem Grund wird der Nachrichtenübersicht mitgeteilt, dass die Zelle ungültig geworden ist und einer Aktualisierung bedarf. Im Anschluss daran wird die Zelle von Thunderbird erneut abgefragt. Der Unterschied besteht darin, dass bei der zweiten Abfrage bereits ein zwischengespeichertes Analyseergebnis in der Datenbank von JackF vorliegt.

```
NS_IMETHODIMP nsJackFCellPositionCallback::Notify()
{
  this->rowArray->RemoveElement(this->row);
  PRInt32 visibleRow;
  nsresult rv = this->tree->GetFirstVisibleRow(&visibleRow);
  if (NS_FAILED(rv)) return rv;
  if (visibleRow <= this->row) {
    rv = this->tree->GetLastVisibleRow(&visibleRow);
    if (NS_FAILED(rv)) return rv;
    if (visibleRow >= this->row) {
      rv = this->tree->InvalidateCell(this->row,this->column);
      if (NS_FAILED(rv)) return rv;
    }
  }
  return NS_OK;
}
```

Abbildung 4.28: Quellcode - nsJackFCellPositionCallback::Notify()

4.5.2 Workerthread Wiederverwendung

Die Funktion `ThreadFinished(…)` von `nsIJackFControlManager` wird von `nsIJackFControlManagerCallback` aufgerufen, nachdem das Analyseergebnis in die Datenbank gespeichert und die Display Klasse benachrichtigt wurde. Die folgende Abbildung zeigt die dafür verantwortlichen Anweisungen. In `ThreadFinished(…)` wird zuerst festgestellt, ob die Warteschlange mit dem Namen `controlManagerObjectStack` leer ist. Ist das der Fall, wird der Workerthread beendet und der Threadzähler um eins vermindert. Ist die Warteschlange gefüllt, wird die nächste Anfrage geladen und der Workerthread weiterverwendet.

```
NS_IMETHODIMP nsJackFControlManager::ThreadFinished(nsIThread
*workerThread)
{
  nsresult rv;
  if (!this->controlManagerObjectStack->empty()) {
    nsCOMPtr<nsIJackFControlManagerObject> controlManagerObject =
      this->controlManagerObjectStack->front();
    this->controlManagerObjectStack->pop_front();
    nsCOMPtr<nsIJackFWorker> jackFWorker =
      do_CreateInstance("@mozilla.org/nsJackFWorker;1", &rv);
    if (NS_SUCCEEDED(rv)){
      rv = jackFWorker->SetCallbackAttributes(controlManagerObject);
      if (NS_SUCCEEDED(rv)){
        nsCOMPtr<nsIRunnable> runnableWorker
          (do_QueryInterface(jackFWorker,&rv));
        if (NS_SUCCEEDED(rv)){
          workerThread->Dispatch
           (runnableWorker,nsIThread::DISPATCH_NORMAL);
        }
      }
    }
  }
  else {
    workerThread->Shutdown();
    this->threadCount--;
  }
  return NS_OK;
}
```

Abbildung 4.29: Quellcode - nsJackFControlManager::ThreadFinished(...)

4.5.3 Zwischenspeicher zum Reduzieren von RDF Manipulationen in nsIJackFDisplayManager

Das Arbeiten mit RDF Dateien ist eine ressourcenintensive Aufgabe, weshalb durch das Aufbewahren bereits aus einer RDF Datei geladener Daten im Hauptspeicher große Performancesteigerungen erzielt werden können. Für das 5.000.000 malige Aufrufen von `GetDisplayExtensionsIDs(…)` ohne Zwischenspeicher werden ungefähr 1.200 Sekunden benötigt. Es mag den Anschein erwecken, dass die 0,24 Millisekunden pro Aufruf so gering sind, dass es kein Einsparungspotenzial gibt. Das ist nicht der Fall, da `GetDisplayExtensionsIDs(…)` immer aufgerufen wird, wenn eine Display Klasse eine Anfrage an `nsIJackFControlManager` stellt. Dies erfolgt beim Scrollen über die zum Testen verwendeten 5.169 Nachrichten durchschnittlich 72.000 Mal, was einer kumulierten Dauer von 17,8 Sekunden entspricht. Durch den Einsatz eines Zwischenspeichers konnte die benötigte Dauer pro Aufruf auf 0,0006 Millisekunden reduziert werden. Dies entspricht einer Senkung des Zeitbedarfs auf 0,25% des ursprünglichen Zeitbedarfs und reduziert die kumulierte Dauer beim Scrollen über die Testdaten auf ungefähr 0,04 Sekunden.

Der in der nachfolgenden Abbildung grün geschriebene Quellcode ist dafür verantwortlich zu bestimmen, ob die benötigte Information bereits einmal aus der RDF Datei geladen wurde und im Zwischenspeicher verfügbar ist. Ist dem nicht so, wird mit dem in blau geschriebenen Quellcode die angeforderte Information mittels Aufruf von `GetDisplayExtension-sIDs(…)` des `nsIJackFDisplaysRDFManager` Services aus der RDF Datei geladen und in den Zwischenspeicher eingefügt.

```
NS_IMETHODIMP nsJackFDisplayManager::GetDisplayExtensionsIDs(const nsAC-
String & displayID, nsTArray<nsCID> * *_retval)
{
   nsresult rv;
   nsTArray<nsCID> *newExtensionCIDs = NULL;
   nsJackFDisplayIDEntry *currentDisplayIDEntry = NULL;
   int i = this->displayCIDsLength;
   while ( (i>=0) && (newExtensionCIDs == NULL) ) {
      currentDisplayIDEntry = this->displayCIDs.ElementAt(i);
      if (currentDisplayIDEntry->displayID == displayID) {
         newExtensionCIDs = currentDisplayIDEntry->extensionIDs;
      }
      else {
         i--;
      }
   }
   if ( i < 0 ) {
      nsCOMPtr<nsIArray> extensionIDs;
      newExtensionCIDs = new nsTArray<nsCID>();
      rv = this->displaysRDFManager->GetDisplayExtensionsIDs(displayID,
         getter_AddRefs(extensionIDs));
      PRUint32 arrayLength;
      rv = extensionIDs->GetLength(&arrayLength);

      for (PRUint32 i = 0; i < arrayLength; ++i) {
         nsCOMPtr<nsISupportsID> extensionIDPR =
            do_QueryElementAt(extensionIDs, i, &rv);
         nsCID *extensionIDCID;
         rv = extensionIDPR->GetData(&extensionIDCID);
         if (NS_FAILED(rv)) return rv;
         newExtensionCIDs->AppendElement(*extensionIDCID);
      }
      nsJackFDisplayIDEntry *newDisplayIDEntry =
         new nsJackFDisplayIDEntry();
      newDisplayIDEntry->displayID = displayID;
      newDisplayIDEntry->extensionIDs = newExtensionCIDs;
      this->displayCIDs.AppendElement(newDisplayIDEntry);
      this->displayCIDsLength = this->displayCIDs.Length()-1;
   }
   *_retval = newExtensionCIDs;
   return NS_OK;
}
```

Abbildung 4.30: Quellcode - nsJackFDisplayManager::GetDisplayExtensionsIDs(…)

4.5.4 nsIJackFScoreDB

Die folgende Abbildung zeigt die Funktion `SetGraphicScore(…)`. Wie bereits erwähnt wird die frei erhältliche SQLite Datenbank verwendet. Die in der Abbildung *4.31: Quellcode - nsJackFScoreDB::SetGraphicScore(...)* grün geschriebenen Bereiche sind für das Initialisieren einer SQL Anweisung, die in SQLite ausgeführt wird, verantwortlich. Mit dem ersten grünen Block wird versucht, ein Analyseergebnis in die Datenbank einzufügen. Ist für die Kombination Nachricht und Pluggy bereits ein Analyseergebnis gespeichert, tritt eine Primärschlüsselverletzung auf, und es wird mit dem zweiten grünen Block ein Update durchgeführt.

SQLite unterstützt zahlreiche performancesteigernde Mechanismen. Aus diesem Grund wird beim Initialisieren von `nsIJackFScoreDB` der maximale Hauptspeicher, der von SQLite benutzt werden darf, festgelegt. Da SQLite Transaktionen unterstützt, muss für einen erfolgreichen Schreibvorgang die Änderung auf der Festplatte abgespeichert werden. Um die Performance von SQLite zu steigern, wurde SQLite von Thunderbird dahingehend optimiert, dass Schreibvorgänge auf die Festplatte im Hintergrund erfolgen, und SQLite während des Schreibvorgangs bereits weiter benutzt werden kann. Transaktionen werden durch diese Performanceoptimierung nur noch bedingt unterstützt. Zwar ist eine Transaktion weiterhin atomar, jedoch ist nach Ausführen eines Commits nicht sichergestellt, dass die Transaktion auch erfolgreich permanent auf die Festplatte geschrieben wurde.

Je mehr kurze Transaktionen auf die Festplatte geschrieben werden müssen, desto aufwendiger ist dies. Unter Umständen kann es vorkommen, dass bei einer großen Anzahl an Änderungen noch auszuführende Schreibvorgänge Thunderbird dran hindern, geschlossen zu werden. Es ist empfehlenswert, möglichst viele Änderungen zu einer größeren Transaktion zusammenzufassen. Aus diesem Grund wird beim Initialisieren von `nsIJackFScoreDB` eine Transaktion gestartet. Mit den in blau gefärbten Zeilen wird nach jeweils 500 Schreibvorgängen die geöffnete Transaktion mit `Commit` beendet und eine neue Transaktion gestartet [Owe06].

```
NS_IMETHODIMP nsJackFScoreDB::SetGraphicScore(const nsACString & msgHash,
const nsCID * extensionID, const nsACString & score, PRTime timestamp){
nsresult rv = mDBConn->CreateStatement(NS_LITERAL_CSTRING("INSERT INTO
  graphicscore VALUES (?1,?2,?3,?4)"),
  geter_AddRefs(insertGraphicStatement));
if (NS_SUCCEEDED(rv)){
  insertGraphicStatement->BindUTF8StringParameter(0, msgHash );
  insertGraphicStatement->BindBlobParameter(1, (PRUint8*) extensionID,
    16);
  insertGraphicStatement->BindUTF8StringParameter(2, score );
  insertGraphicStatement->BindInt64Parameter(3, timestamp );
  rv = insertGraphicStatement->Execute();
  if (NS_FAILED(rv)){
    PRInt32 lastError;
    rv = mDBConn->GetLastError(&lastError);
    insertGraphicStatement->Reset();
    if (lastError == SQLITE_CONSTRAINT) {
      insertGraphicStatement->Reset();
      rv = mDBConn->CreateStatement(NS_LITERAL_CSTRING("UPDATE
        graphicscore SET score = ?3,timestamp = ?4 WHERE msgHash=?1 and
        extensionID = ?2"), getter_AddRefs(updateGraphicStatement));
      if (NS_FAILED(rv)) return NS_ERROR_FAILURE;
      updateGraphicStatement->BindUTF8StringParameter(0, msgHash );
      updateGraphicStatement->BindBlobParameter(1, (PRUint8*)
        extensionID, 16 );
      updateGraphicStatement->BindUTF8StringParameter(2, score );
      updateGraphicStatement->BindInt64Parameter(3, timestamp );
      rv = updateGraphicStatement->Execute();
      updateGraphicStatement->Reset();
      if (NS_FAILED(rv)) return NS_ERROR_FAILURE;
    }
  }
  else {
    insertGraphicStatement->Reset();
  }
  this->operationCount++;
  if (this->operationCount == 512) {
    this->operationCount = 0;
    this->mDBConn->CommitTransaction();
    this->mDBConn->BeginTransaction();
  }
}
return NS_OK;
}
```

Abbildung 4.31: Quellcode - nsJackFScoreDB::SetGraphicScore(…)

4.6 Schreiben eines Pluggies – Entwicklerhandbuch

Für das Schreiben eines Pluggies wird eine Vorlage zur Verfügung gestellt, welche vom Entwickler an das gewünschte Pluggy angepasst werden kann. Die sechs dafür notwendigen Schritte werden in der nun folgenden Anleitung beschrieben.

1. Einen Ordner mit dem Namen des zu schreibenden Pluggys anlegen.
2. Den Inhalt von `jackftemplate.zip` in den Ordner von Schritt eins entpacken. Bei `jackftemplate.zip` handelt es sich um alle, zum Entwickeln eines in JS geschrie-

ben Pluggies, benötigten Dateien. Das Einrichten einer Entwicklungsumgebung, wie in Kapitel *3.2 Environment für einfache Projekte* beschrieben wird, ist nicht notwendig.

3. Öffnen der Datei `install.rdf` und abändern der darin gespeicherten Werte.

   a. `<em:id>{1DDADE37-1788-4956-920E-C905C9C7DE7E}</em:id>` muss durch eine neue eindeutige ID[a] ersetzt werden.

   b. `<em:name>Jack-F Template</em:name>`, `<em:description>This is the Template example!</em:description>`, `<em:version> 1.0</em:version>`, `<em:creator>Christian Haider</em:creator>` repräsentieren den Namen des Pluggies, eine Beschreibung, eine Versionsnummer und den Namen des Autors. Diese Angaben müssen entsprechend der Wünsche des Pluggy-Autors angepasst werden.

   c. `<em:homepageURL>http://localhost/ChristianChristian</em:homepageURL>` Sollte der Pluggy-Autor einen Verweis auf seine Webseite wünschen, kann er mit diesem Wert eine Verknüpfung setzen. Sollte kein Verweis gewünscht sein, muss diese Zeile gelöscht werden.

   d. `<em:iconURL>chrome://jackftemplate/content/icon.png</em:iconURL>` ist eine CHROME URL zu jenem Icon, welches beim Installieren und in der Liste unter `Add-ons` von Thunderbird angezeigt wird. „`jackftemplate`" ist mit dem Namen des neuen Pluggys zu ersetzen. Dieser sollte mit dem Namen des unter Punkt 1 angelegten Ordners übereinstimmen.

   e. `<em:updateURL>http://localhost/update.rdf</em:updateURL>` ermöglicht es, eine Updateadresse anzugeben. Wenn das Pluggy kein automatisches Update unterstützen soll, muss diese Zeile gelöscht werden.

4. Öffnen der Datei `chrome.manifest` und ändern der Zeichenkette „`jackftemplate`" auf den Namen des Pluggys. Dieser sollte mit dem Namen des unter Punkt 1 angelegten Ordners übereinstimmen.
5. Ersetzen der Datei `icon.png` im `content` Ordner mit der Logo Datei des Pluggys.
6. Umbenennen der Datei `jackftemplate.js` im `components` Ordner auf den Namen des Pluggys. Dieser sollte mit dem Namen des unter Punkt 1 angelegten Ordners übereinstimmen.

[a] http://www.guidgen.com/

Die Vorlage ist nun an das neue Pluggy angepasst und kann in JackF eingebunden werden. Das Pluggy würde ohne weitere Anpassung grafische, numerische und alphanumerische Analysen der „wirklichen“ Absendezeit durchführen. Um dem Pluggy die eigentlich gewünschte Funktionalität zu geben, muss nun damit begonnen werden, die in Schritt 7 umbenannte Datei zu editieren. Es sind in Summe 9 Konstanten zu ändern und an 3 Positionen die unerwünschte Funktionalität zu löschen. Die Vorlage ist so aufgebaut, dass ein Pluggy mit höchstem Funktionsumfang vorhanden ist, und dieser vom Entwickler reduziert werden muss.

```
const COMPONENT_CONTRACTID = '@mozilla.org/jackfjackftemplate;1';
const COMPONENT_CID = Components.ID('{500F8DF4-A41E-41f9-B22C-
  A063ED731C66}');
const COMPONENT_CLASSNAME = 'jackfjackftemplate';
const _logo = "chrome://jackftemplate/content/pics/icon.png";
const _version = "1.0";
const _name = "JackFTemplate";
const _description = "This is the template for all new Jack-F Pluggies.";
const _timeout = 30;
const _cachetime = 10;
```

Abbildung 4.32: Quellcode – Konstanten von nsIJackFExtension in jackftemplate.js

Obige Abbildung zeigt die zu ändernden Konstanten. Diese sind wie folgt zu wählen. `COMPONENT_CONTRACTID` ist der Name, unter dem ein Objekt des Pluggies angelegt werden kann. Dieser sollte für das Pluggy sprechend und eindeutig sein. `COMPONENT_CID` ist eine eindeutige ID[a] des Pluggys. `COMPONENT_CLASSNAME` ist der Klassenname des Pluggys, der folglich auch für das Pluggy sprechen sollte. `_logo`, `_version` `_name` und `_description` sind Konstanten, welche verwendet werden, um das Pluggy in JackF zu registrieren. Die hier eingestellten Werte werden im Konfigurationsmenü von JackF angezeigt. `_timeout` wird in Millisekunden angegeben und steuert die maximal verfügbare Zeit, die das Pluggy zum Analysieren einer Nachricht benötigt. Nach Ablauf dieser Zeitspanne wird das Pluggy von JackF beendet. Mit `_cachetime` ist es möglich, den Zeitraum anzugeben, in dem ein gespeichertes Analyseergebnis gültig ist. Nach Ablauf dieser in Sekunden angegebenen Zeitspanne muss bei Bedarf erneut eine Analyse durchgeführt werden. Wird -1 als Zeitspanne gewählt, ist das Analyseergebnis endlos gültig.

In den Zeilen 34-36, 53-55 und 271-273 sind jene Zeilen zu löschen, die angeben, dass ein Pluggy eine entsprechende Analyseform unterstützt. Sollte ein Pluggy etwa keine numerische Analyse unterstützen, sind die Zeilen 34, 53 und 271 zu löschen. Die Angabe, welche Zeile für welche Analyseform zuständig ist, ist als Kommentar in der jeweiligen Zeile angemerkt.

[a] http://www.guidgen.com/

Sind nun alle Konstanten gesetzt und die gewünschten Analysefunktionen gewählt, kann in den Funktionen `analyseNumeric: function(msg)`, `analyseGraphic: function(msg)` und `analyseAlphanumeric: function(msg)` die gewünschte Funktionalität implementiert werden. Wahlweise können noch die Funktionen `convertToGraphic: function(score)` und `convertToAlphanumeric: function(score)` zur Performancesteigerung implementiert werden.

Vom nun fertig programmierten Pluggy kann mittels Starten der im `jackftemplate.zip` befindlichen `make.bat` Datei ein installierbares .xpi Paket erstellt werden.

4.7 Probleme

In diesem Kapitel werden die während der Entwicklung von JackF aufgetretenen Probleme beschrieben. Diese reichen von einfachen Hindernissen, wie Typunverträglichkeiten zwischen XPCOM und C++, über Änderungen an der API von Thunderbird, die bei Releasewechsel aufgetreten sind, bis zu einigen tiefgreifenden Fehlern im Quellcode von Thunderbird. Entsprechend der Kategorien wird das Kapitel in die beiden Unterkapitel Bugs im Mozilla Projekt und Releasewechsel von Thunderbird gegliedert.

4.7.1 Bugs im Mozilla Projekt

Mit dem Begriff Bugs wird in diesem Kapitel auf Fehler in Thunderbirds Quellcode oder dessen Build-Prozess aufmerksam gemacht. Da sich im Umfeld der Entwicklung von Thunderbird der Ausdruck Bug etabliert hat, wird darauf verzichtet, diesen mit Fehler zu übersetzen.

Nicht alle der in diesem Kapitel beschriebenen Bugs haben eine Auswirkung auf die gewöhnliche Nutzung von Thunderbird sondern treten ausschließlich in bestimmten Situationen auf. Bugs werden nach deren Dringlichkeit gereiht und danach abgearbeitet. Aus diesem Grund ist es nicht möglich, eine Vorhersage zu treffen, ob und wann diese behoben werden. So weit möglich wurde versucht einen Weg zu finden, trotz diverser Bugs die volle Funktionalität von JackF zu gewährleisten.

Die Überschriften Bug 406414 und ähnliche beruhen darauf, dass für jeden gemeldeten Bug eine aufsteigende Nummer vergeben wird. Es ist dadurch möglich, unter https://bugzilla.mozilla.org/show_bug.cgi?id= durch Anstellen der Nummer

des Bugs die vollständige Geschichte des Bugs zu lesen. Diese öffentlich zugängliche Plattform wird Bugzilla[a] genannt und soll Entwicklern durch das Melden von Problemen durch Anwender dabei helfen, Bugs schneller zu beheben. Um die Entwickler beim Abarbeiten der gemeldeten Bugs zu entlasten, wird von Bugzilla einige Vorarbeit durch den Anwender gefordert. Zum Beispiel gibt es eine Liste von häufig gemeldeten Bugs, welche vom Anwender zu durchsuchen ist, um zu vermeiden, dass Probleme mehrfach als Bugs in Bugzilla eingetragen werden [AMH05].

4.7.1.1 Bug 406414

Bei diesem Bug handelte es sich um ein Problem im Build-Prozess. Durch ihn war es nicht möglich, selbst entwickelte Erweiterungen, welche Quellcodedateien von Thunderbird einbinden, in einem Durchgang mit Thunderbird zu kompilieren. Es war möglich, dieses Problem durch Kompilieren von Thunderbird in einem ersten Schritt und der Erweiterung in einem Zweiten zu umgehen. Dies ließ den Schluss zu, dass das Problem auf der Reihenfolge, in welcher der Build-Prozess die Quelldateien abarbeitet, beruht. Durch richtiges Einstellen des Build-Prozesses, sodass zuerst Thunderbird und im Anschluss die zusätzlichen Erweiterungen kompiliert werden, wurde dieser Bug behoben.

4.7.1.2 Bug 400627

Dieser Bug trat das erste Mal in Thunderbird 2.0.0.7pre auf und ist in der aktuellen Version Thunderbird 3.0a1pre nicht behoben. Dieser Bug äußert sich, wenn 2 Components in einer seltenen Konstellation verwendet werden: Zuerst wird dabei ein Objekt einer in JS entwickelten Component angelegt, das im Anschluss für das Verwenden in mehreren Threads vorbereitet wird. Im Anschluss muss ein Objekt der Component `@mozilla.org/net-work/sync-stream-listener;1` angelegt und verwendet werden.

Das Fehlverhalten des Bugs hat sich in den Versionen von Thunderbird geändert. Während das Ergebnis in Thunderbird 2.0.0.7pre ein unerwartetes Beenden von Thunderbird war, wird in Thunderbird 3.0.a1pre ein endloser rekursiver Aufruf einer Funktion gemeldet. Da der Bug sehr speziell ist, kann nicht auf eine baldige Korrektur gehofft werden. Durch eine Umstellung in JackF hat dieser Bug für das JackF Projekt keine Relevanz mehr.

[a] http://www.bugzilla.org/

4.7.1.3 Bug 419192

Der Bug 419192 stellt ein ernsthaftes Problem für JackF dar, da er einen wichtigen Punkt der Kernfunktionalität von JackF unbrauchbar macht. Durch ihn ist es nicht möglich, eine in JS geschriebene Component innerhalb eines Threads, welcher nicht der `MainThread` ist, zu benutzen. Das Resultat ist, dass JackF keine Pluggies unter der Verwendung von Threads abarbeiten kann, welche in JS geschrieben wurden. Das Resultat wäre, dass entweder die Benutzeroberfläche während der Analyse durch in JS geschriebene Pluggies einfrieren würde, oder JackF keine in JS geschriebenen Pluggies unterstützen könnte. Leider ist der Bug noch nicht behoben, obwohl bereits daran gearbeitet wird. Bug 421877 ist eine Fortsetzung des Bugs 419192, in welchem das Problem bereits genauer lokalisiert wurde. Es hat den Anschein, dass beim Anlegen einer Component die dazu benutzte Factory einen Fehler verursacht, da diese im `MainThread` angelegt wurde und von einem anderen Thread benutzt wird. Es bleibt abzuwarten, ob der Bug bis zum Erscheinen von Thunderbird 3.0 behoben sein wird.

4.7.2 Releasewechsel von Thunderbird

Während der Entwicklung von JackF erfolgten zwei Releasewechsel, dies führte beide Male zu einer Anpassung von JackF an die neuen Begebenheiten. Bei einem Releasewechsel werden nicht nur einschneidende Änderungen an der für den Endbenutzer merklichen Funktionalität oder an der API vorgenommen, sondern auch striktere Prüfungen von Erweiterungen eingeführt. Dadurch wird versucht, Entwicklern eine Hilfestellung bezüglich der frühzeitigen Erkennung oder gänzlichen Vermeidung von Problemen zu geben.

4.7.2.1 Version 1.5 → 2.0

Die Umstellung von Thunderbird 1.5 auf Thunderbird 2.0 erfolgte in einer frühen Phase der Entwicklung von JackF, weshalb keine entscheidenden Änderungen vorgenommen werden mussten.

Für JackF waren zwei Erneuerungen relevant. Zum einen wurde die Möglichkeit geschaffen, zusätzliche Spalten mittels Overlay in der Nachrichtenübersicht einzufügen. Diese wurde vorerst aufgegriffen, musste allerdings zu einem späteren Zeitpunkt verworfen werden. Mittels Overlay könnte lediglich eine fix festgelegte Anzahl an Spalten eingefügt werden. Da JackF

vom Anwender einstellbar ist, müssen auch die Spalten dynamisch eingefügt werden können. Aus diesem Grund wurde mittels JS auf eine andere Vorgehensweise ausgewichen.

Die zweite Änderung ermöglichte es, ab Thunderbird 2.0 zu prüfen, ob von einer Component, welche als Service benutzt wird, ein neu erstelltes Objekt angefordert wird. Ist das der Fall, widerspricht dies dem Konzept, dass von einem Service ausschließlich ein Objekt existieren darf. JackF hätte mehrere Objekte eines Services erzeugt und hätte damit gegen dieses Konzept verstoßen. Dank dieser zusätzlichen Prüfung konnte der Fehler frühzeitig erkannt und behoben werden.

4.7.2.2 Version 2 → 3

Der Umstieg von Thunderbird 2.0.0.8 auf Thunderbird 3.0.a1pre erwies sich als notwendig, da es durch Bug 400627 nicht möglich war, eine Prüfsumme einer Nachricht zu berechnen. Durch die niedrige Priorität des Bugs und das zielstrebige Arbeiten an Thunderbird 3.0 war abzusehen, dass der Fehler in Thunderbird 2.x nicht mehr behoben wird. Zu diesem Zeitpunkt war JackF etwa zu 90% fertig gestellt und gänzlich in JS implementiert. Durch den Releasewechsel auf Thunderbird 3.0a1pre war es notwendig, JackF zur Gänze neu zu implementieren. Lediglich die Benutzeroberfläche, das Design und einige wenige Components sind von JackF für Thunderbird 2.x übernommen worden. Die Gründe dafür sowie die vorgenommenen Anpassungen werden im folgenden näher erläutert.

Umstellung von JS auf C++: Da JackF mit einer Vielzahl an Threads arbeitet, müssen diese untereinander synchronisiert werden. Es ist zum Beispiel notwendig, dass die Warteschlange, in der Anfragen abgelegt werden, die auf ihre Abarbeitung warten, vor gleichzeitigem Schreiben und Lesen verschiedener Threads geschützt wird. Zu diesem Zweck kann in JS nur eine boolesche Variable herangezogen werden, welche angibt, ob die Warteschlange beschrieben werden darf oder nicht. Durch diese Vorgehensweise kann eine Art Semaphor simuliert werden. Da JS keine Möglichkeit bietet, einen Abschnitt als ununterbrechbar zu kennzeichnen, birgt auch diese Lösung ein gewisses Restrisiko. Unter Umständen kann es vorkommen, dass ein Thread prüft, ob die Warteschlange verändert werden darf. Dieser wird nach der Prüfung, bevor er die boolesche Variable setzt, unterbrochen. Wenn nun auch ein zweiter Thread prüft, ob die Warteschlange verändert werden darf, kann es vorkommen, dass beide Threads gleichzeitig den kritischen Bereich abarbeiten. Das Restrisiko des oben beschriebenen Szenarios

kann nicht gänzlich ausgeschlossen werden, es konnte allerdings durch einen 4stündigen Dauertest, der auf die Provokation dieses Szenarios ausgelegt war, kein Fehlverhalten von JackF festgestellt werden. Das Problem bei simulierten Semaphoren ist es, dass diese bei Auftreten einer Sperre einen Thread für einen bestimmten Zeitraum pausieren lassen müssen. Dies konnte in Thunderbird 2.0 durch Aufruf einer Funktion in JS bewerkstelligt werden. Diese Möglichkeit wurde in Thunderbird 3.0a1pre ersatzlos gestrichen, was dazu führte, dass es nicht mehr möglich ist, einen Thread in JS pausieren zu lassen. Das Ergebnis unsauberer Lösungen, die nutzlose Operationen auf Kosten anderer Anwendungen ausführen, ist die Vergeudung von Rechenzeit. Durch eine Portierung von JS nach C++ gab es sowohl die Möglichkeit echte Semaphoren zu nutzen als auch Threads pausieren zu lassen.

Entfernen von zusätzlichen Bibliotheken: JackF für Thunderbird 2.0 setzte eine frei verfügbare Bibliothek namens jsLib[a] ein. Diese Bibliothek stellte eine Reihe nützlicher Funktionen in JS zur Verfügung. Unter anderem war es mit Hilfe von jsLib möglich, Dateien aus komprimierten Dateien zu extrahieren. Beim Starten von JackF überprüfen die Components `nsIJackFDisplaysRDFManager` und `nsIJackFExtensionsRDFManager`, ob die zum Abspeichern der Einstellungen benötigten RDF Dateien im Verzeichnis von JackF angelegt sind. Waren diese nicht vorhanden, wurde jsLib verwendet, um aus dem Installationspaket von JackF die benötigten Dateien zu extrahiere und an gewünschter Stelle abzulegen. Da keine jsLib Version für Thunderbird 3 zur Verfügung stand, wurde auf dessen Verwendung verzichtet. Bei Bedarf werden die RDF Dateien auf anderem Wege erzeugt.

Benutzung von Threads: Das Anlegen und Benutzen von Threads, sowie die Kommunikation zwischen diesen wurde beim Wechsel auf Thunderbird 3 gänzlich geändert. Während in Thunderbird 2.0 die Kommunikation mittels speziell darauf vorbereiteter Objekte einer Component erfolgen musste, wird das in Thunderbird 3 durch das Senden von einer Art Signal realisiert. Dies hat den Vorteil, dass die Handhabung strikter und dadurch der Quellcode fehlerfreier wird.

[a] http://jslib.mozdev.org/

5 Zusammenfassung

Durch die Implementierung des JackF Frameworks konnte gezeigt werden, dass die Strukturschwäche von Thunderbird, welche das Einbinden von Erweiterungen erschwert, auf abstrakter Ebene effizient gelöst werden kann. Es war möglich, Nebenaufgaben einer Analyse für Entwickler komfortabel durch das Framework erledigen, zu lassen, wodurch der Fokus eines Entwicklers auf die eigentliche Analysefunktion gelenkt wird.

Die geplante Implementierung von JackF in JS konnte nicht realisiert werden, sondern es musste ein Kompromiss zwischen Performance, Stabilität und Plattformunabhängigkeit getroffen werden. Die Implementierung wurde Größtenteils in C++ vorgenommen und mit Einbindung in den Mozilla Build-Prozess eine quasi Plattformunabhängigkeit erreicht.

Eine Schwäche von JackF ist es, dass die grafische Darstellung von Analyseergebnissen in der Nachrichtenübersicht für jedes Pluggy eine eigene Spalte benötigt. Es ist nicht möglich, mehrere Grafiken in einer Spalte nebeneinander darzustellen. Eine Lösung könnte sein, alle anzuzeigenden Grafiken on-the-fly zu einer gemeinsamen Grafik zu vereinen. Da dies eine ressourcenintensive Aufgabe wäre, wird dies jedoch nicht ohne Performanceeinbußen realisierbar sein.

Die erste Bewährungsprobe für JackF ist es, sich in der Sandbox für Thunderbird Erweiterungen zu beweisen. Mit zunehmender Bekanntheit von JackF werden Entwickler Pluggies für JackF implementieren, was die Verbreitung weiter fördern wird. Das Problem, das dabei ersichtlich wird, ist, dass ohne entsprechende Bekanntheit keine Pluggies entwickelt werden, und umgekehrt ohne Pluggies die Bekanntheit nicht zunimmt. Als Lösung für dieses Dilemma könnten alle Erweiterungen, die in Kapitel *2.3 Bestehende Lösungen* genannt wurden, als Pluggy implementiert werden und somit eine Grundmenge verfügbaren Pluggies geschaffen werden.

Die Funktionen, die ein Framework wie JackF bieten kann, sind noch nicht vollständig ausgereizt. Daher ist es geplant, JackF nach einer ersten Testphase mit zusätzlicher Funktionalität zu erweitern. So ist geplant, die Kombination von verschiedenen Pluggies zu ermöglichen, wodurch zum Beispiel Durchschnittswerte von numerischen Pluggies ermöglicht werden. Weiters ist geplant, eine Exportmöglichkeit für die in JackF vorgenommen Einstellungen zu realisieren. Zu den exportierbaren Einstellungen zählen die höchstzulässige Anzahl gleichzei-

tig durchgeführter Analysen, sämtliche Pluggies und deren Timouts und die Zeitdauer, in welcher deren Analyseergebnisse gültig sind. Zusätzlich zu den Einstellungen der Pluggies sollen die Einstellungen der Displays exportierbar sein. Damit wäre es möglich, maßgeschneiderte Displays, welche verschiedene Pluggies kombinieren, zur Verfügung zu stellen und automatisch alle dafür benötigten Pluggies bei Bedarf aus dem Internet nachzuladen und zu installieren.

Zusätzliche Ausgabemöglichkeiten könnten den Einsatzbereich von JackF weiter verbreiten. Neben weiteren bildhaften Ausgabemöglichkeiten wäre es möglich, zum Beispiel akustische Signale zu verwenden, wenn eine Nachricht besonderes Gefahrenpotenzial birgt. Dadurch könnte ein hilfreiches Werkzeug für sehbehinderte Personen geschaffen werden.

6 Abbildungs- und Tabellenverzeichnis

ABBILDUNG 2.1: THUNDERBIRDS BENUTZEROBERFLÄCHE 10
ABBILDUNG 2.2: ADD-ONS FENSTER 11
ABBILDUNG 2.3: DISPLAY MAIL USER AGENT BEISPIEL 12
ABBILDUNG 2.4: COUNTRY LOOKUP BEISPIEL 13
ABBILDUNG 2.5: IRANZILLA BEISPIEL 13
ABBILDUNG 2.6: MAILCLASSIFIER BEISPIEL 14
ABBILDUNG 2.7: SENDER VERIFICATION ANTI-PHISHING EXTENSION 14
ABBILDUNG 2.8: SHOW INOUT BEISPIEL 15
ABBILDUNG 2.9: SENDERFACE BEISPIEL 15
ABBILDUNG 2.10: MESSAGE LEVEL AUTHENTICATION BEISPIEL 15
ABBILDUNG 2.11: SPAMNESS BEISPIEL 16
ABBILDUNG 2.12: ENIGMAIL BEISPIEL 16
ABBILDUNG 3.1: STRUKTUR EINER .XPI DATEI 23
ABBILDUNG 3.2: XUL BEISPIEL AUSGABE 25
ABBILDUNG 3.3: XUL BEISPIEL ALS DOM 26
ABBILDUNG 3.4: OVERLAY 27
ABBILDUNG 3.5: MENÜLEISTE ALS OVERLAY 28
ABBILDUNG 3.6: CONTENTS.RDF OVERLAY AUFLISTUNG 28
ABBILDUNG 3.7: BEISPIEL OVERLAY QUELLCODE 29
ABBILDUNG 3.8: BEISPIEL OVERLAY AUSGABE 29
ABBILDUNG 3.9: BEISPIEL EXAMPLE.DTD UND EXAMPLE.XUL DATEI 31
ABBILDUNG 3.10: PREFERENCE WINDOW VON THUNDERBIRD 32
ABBILDUNG 3.11: BEISPIEL VERWENDUNG VON PREFERENCES IN EXAMPLE.XUL 33
ABBILDUNG 3.12: BEISPIEL INSTALL.RDF 35
ABBILDUNG 3.13: BEISPIEL CHROME.MANIFEST 35
ABBILDUNG 3.14: .XPI PAKET 37
ABBILDUNG 3.15: BEISPIEL .IDL DATEI 42
ABBILDUNG 3.16: PEEL VIEW OF XPCOM COMPONENT CREATION [TUOE03] 43
ABBILDUNG 3.17: .MOZCONFIG DATEI 48
ABBILDUNG 3.18: .MOZCONFIG DATEI MIT ERWEITERUNG 50
ABBILDUNG 3.19: MAKEDATEI IM HAUPTVERZEICHNIS DER BEISPIELERWEITERUNG 51
ABBILDUNG 3.20: MAKEDATEI IM SRC VERZEICHNIS DER BEISPIELERWEITERUNG 52
ABBILDUNG 3.21: MAKEDATEI IM PUBLIC VERZEICHNIS DER BEISPIELERWEITERUNG 53
ABBILDUNG 4.1: ADD-ONS KONFIGURATION VON THUNDERBIRD (LI.) UND JACKF (RE.) 57
ABBILDUNG 4.2: ANALYSEERGEBNIS IN SPALTE AUSGEGEBEN 58
ABBILDUNG 4.3: ANALYSEERGEBNIS NEBEN KOPFDATEN AUSGEGEBEN 58
ABBILDUNG 4.4: JACKF PLUGGY-INTERFACES 60
ABBILDUNG 4.5: SCHEMATISCHE DARSTELLUNG EINER ANFRAGE 63
ABBILDUNG 4.6: SCHEMATISCHE DARSTELLUNG VON JACKF 66
ABBILDUNG 4.7: KONFIGURATIONSMENÜ JACKF 69
ABBILDUNG 4.8: KONFIGURATION EINES DISPLAYS 70
ABBILDUNG 4.9: KONFIGURATION EINES PLUGGIES 71
ABBILDUNG 4.10: INTERFACE MIT KLASSE UND XPCOM IN C++ 73
ABBILDUNG 4.11: TRENNUNG VON PRÄSENTATIONSSCHICHT UND ANWENDUNGSLOGIK 73
ABBILDUNG 4.12: KLASSENDIAGRAMM - DISPLAY KLASSEN 74
ABBILDUNG 4.13: SEQUENZDIAGRAMM - EINFACHE ANFRAGE MIT ZWISCHENGESPEICHERTEM ANALYSEERGEBNIS 75
ABBILDUNG 4.14: SEQUENZDIAGRAMM - ANFRAGE OHNE ZWISCHENGESPEICHERTEM ANALYSEERGEBNIS 76
ABBILDUNG 4.15: KLASSENDIAGRAMM - DATENBANK KLASSEN 77
ABBILDUNG 4.16: SEQUENZDIAGRAMM - UMRECHNUNG EINES NUMERISCHEN ANALYSEERGEBNISSES IN EIN GRAFISCHES 77
ABBILDUNG 4.17: SEQUENZDIAGRAMM - SPEICHERN EINES ANALYSEERGEBNISSES 79
ABBILDUNG 4.18: KLASSENDIAGRAMM - KONFIGURATIONSKLASSEN EXTENSIONMANAGER 80
ABBILDUNG 4.19: ZWISCHENSPEICHER ZUR PERFORMANCESTEIGERUNG VON RDF DATENHALTUNG 80
ABBILDUNG 4.20: KLASSENDIAGRAMM - KONFIGURATIONSKLASSEN DISPLAYMANAGER 81
ABBILDUNG 4.21: KLASSENDIAGRAMM – JACKF KERNSTÜCK 82
ABBILDUNG 4.22: SEQUENZDIAGRAMM – JACKF KERNSTÜCK 83
ABBILDUNG 4.23: KLASSENDIAGRAMM – WORKER KLASSEN 84
ABBILDUNG 4.24: SEQUENZDIAGRAMM - KOMMUNIKATION THREAD MIT MAINTHREAD 85

ABBILDUNG 4.25: KLASSENDIAGRAMM - HILFSKLASSEN 86
ABBILDUNG 4.26: SEQUENZDIAGRAMM – PLUGGY REGISTRIERUNG 87
ABBILDUNG 4.27: SEQUENZDIAGRAMM – ANALYSEABFRAGE AUF HÖCHSTEM DETAILLIERUNGSGRAD 88
ABBILDUNG 4.28: QUELLCODE - NSJACKFCELLPOSITIONCALLBACK::NOTIFY() 92
ABBILDUNG 4.29: QUELLCODE - NSJACKFCONTROLMANAGER::THREADFINISHED(...) 93
ABBILDUNG 4.30: QUELLCODE - NSJACKFDISPLAYMANAGER::GETDISPLAYEXTENSIONSIDS(…) 94
ABBILDUNG 4.31: QUELLCODE - NSJACKFSCOREDB::SETGRAPHICSCORE(…) 96
ABBILDUNG 4.32: QUELLCODE – KONSTANTEN VON NSIJACKFEXTENSION IN JACKFTEMPLATE.JS 98

TABELLE 4.1: PERFORMANCEVERGLEICH ZWEIER JACKF VERSIONEN 62

7 Literaturverzeichnis

[BCKMO02] DAVID, BOSWELL; PETE, COLLINS; BRIAN, KING; ERIC, MURPHY; IAN, OESCHGER: *Creating Applications with Mozilla.* O'Reilly & Associates, Inc. Sebastopol – USA, 2002. ISBN: 0-596-00052-9.

[BNSO08] LEN, BASS; ROBERT L., NORD; RAGHVINDER S., SANGWAN; IPEK, OZKAYA: *Making Practical Use of Quality Attribute Information.* In: IEEE Software March 2008, Seiten 24-33. 2008.

[Cha01] Pierre-Antoine, Champin: RDF Tutorial. 2001. URL: http://www710.univ-lyon1.fr/~champin/rdf-tutorial/rdf-tutorial.pdf (letzter Aufruf: 23.04.08)

[DaMu01] JAMES R., DABROWSKI; ETHAN V., MUNSON: *Is 100 Milliseconds Too Fast?* In: Conference on Human Factors in Computing Systems CHI '01 extended abstracts on Human factors in computing systems, Seiten 317-318. ACM New York – USA, 2001. ISBN:1-58113-340-5.

[Emm97] WOLFGANG, EMMERICH: *An introduction to OMG/CORBA*. In: International Conference on Software Engineering Proceedings of the 19th international conference on Software engineering, Seiten 641-642. ACM New York – USA, 1997. ISBN: 0-89791-914-9.

[KrPo88] GLENN E., KRASNER; STEPHEN T., POPE: *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80.* In: Journal of Object-Oriented Programming Volume 1; Issue 3, Seiten 26-49. SIGS Publications Denville – USA, 1988. ISSN: 0896-8438.

[LWW05] ZHANGXI, LIN;TONGSEN, WANG; LEI, WU: *The Revival of Mozilla in the Browser War Against Internet Explorer.* In: ACM International Conference Proceeding Series; Vol. 113 Proceedings of the 7th international conference on Electronic commerce, Seiten 159-166. ACM New York – USA, 2005. ISBN: 1-59593-112-0.

[Mec04] ROBERT, MECKLENBURG: *Managing Projects with GNU Make – The Power of GNU Make for Building Anything.* O'Reilly Media, Inc., 3 edition 2004. ISBN: 0-596-00610-1.

[Owe06] MICHAEL, OWENS: *The Definitive Guide to SQLite.* Springer-Verlag New York, Inc. – USA, 2006. ISBN: 1-59059-673-0.

[Rel03] RELEVANTIVE AG: *Linux Usability Studie - Version 1.0.1.*. 2003. URL: http://www.linux-usability.de/download/linux_usability_report.pdf (letzter Aufruf: 23.04.08).

[Ste07] BRENT, STEARN: *XULRunner: A New Approach for Developing Rich Internet Applications.* In: IEEE Internet Computing Volume 11; Issue 3, Seiten 67-73. IEEE Educational Activities Department Piscataway – USA, 2007. ISSN: 1089-7801.

[Sun05] SUNBELT SOFTWARE - SPYWARE RESEARCH CENTER: *Spyware, Adware, & Mozilla Firefox.* URL: http://research.sunbelt-software.com/Spyware_Adware_Mozilla.pdf (letzter Aufruf: 23.04.08)

[TuOe03] DOUG, TURNER; IAN, OESCHGER: *Creating XPCOM Components.* Brownhen Publishing, 2003. URL: http://www.mozilla.org/projects/xpcom/book/cxc/ (letzter Aufruf: 23.04.08).

8 Appendix

8.1 Quellcode Example

8.1.1 contents.rdf (Seite 28)

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <RDF:Seq about="urn:mozilla:package:root">
    <RDF:li resource="urn:mozilla:package:example"/>
  </RDF:Seq>

  <RDF:Seq RDF:about="urn:mozilla:overlays">
    <RDF:li resource="chrome://messenger/content/messenger.xul"/>
  </RDF:Seq>

  <RDF:Seq about="chrome://messenger/content/messenger.xul">
    <RDF:li>chrome://example/content/overlay.xul</RDF:li>
  </RDF:Seq>

  <RDF:Description about="urn:mozilla:package:example"
    chrome:displayName="Example"
    chrome:author="Christian Haider"
    chrome:authorURL="mailto:christian1111@aon.at"
    chrome:name="example"
    chrome:extension="true"
    chrome:description="This is an extension example!">
  </RDF:Description>
</RDF:RDF>
```

8.1.2 example.xul (Seite 31 und 33)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<!DOCTYPE windows SYSTEM "chrome://example/locale/example.dtd">
<window xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
    title="&example.label;" class="dialog">
  <preferences>
    <preference id="pref_name" name="example.name" type="string"/>
  </preferences>
  <groupbox align="center" orient="horizontal">
    <vbox>
      <text value="&example.label;" style="font-weight: bold; font-
        size: x-large;"/>
      <text value="ver. 1.0"/>
      <separator class="thin"/>
      <hbox>
        <text value="&createdby.label;" style="font-weight: bold;"/>
        <text value="Christian Haider"/>
      </hbox>
      <hbox>
```

```
          <label value="&name.label;"/>
          <textbox id="name" preftype="char" flex="1" preference=
          "pref_name"/>
        </hbox>
        <spacer flex="1"/>
        <button label="&close.label;" accesskey="&close.key;" oncommand=
          "window.close();"/>
      </vbox>
  </groupbox>
</window>
```

8.1.3 JS Beispiel Component

```
const MYCOMPONENT_CLASSNAME = 'nsExample1';
const MYCOMPONENT_CONTRACTID = '@mozilla.org/example1;1';
const MYCOMPONENT_CID = Components.ID('{C94D7DC2-42F5-4459-A804-
  E4C3FB3AC4FA}');
const MYCOMPONENT_IID = Components.interfaces.nsIExample;

function nsExample() { }

nsExample.prototype = {
  allCapital: function(s) {
  return s.toUpperCase();
  },

  QueryInterface: function(iid) {
    if (iid.equals(Components.interfaces.nsISupports) ||
      iid.equals(MYCOMPONENT_IID))
      return this;
    throw Components.results.NS_ERROR_NO_INTERFACE;
  }
};

var nsExampleModule = {
  registerSelf: function(compMgr, fileSpec, location, type) {
    compMgr = compMgr.QueryInterface
            (Components.interfaces.nsIComponentRegistrar);
    compMgr.registerFactoryLocation(MYCOMPONENT_CID,MYCOMPONENT_CLASSNAME,
      MYCOMPONENT_CONTRACTID, fileSpec, location, type);
  },

  getClassObject: function(compMgr, cid, iid) {
    if (!cid.equals(MYCOMPONENT_CID))
      throw Components.results.NS_ERROR_NO_INTERFACE;
    if (!iid.equals(Components.interfaces.nsIFactory))
      throw Components.results.NS_ERROR_NOT_IMPLEMENTED;
    return nsExampleFactory;
  },

  canUnload: function(compMgr) { return true; }
};

var nsExampleFactory = {
  createInstance: function(outer, iid) {
    if (outer != null)
      throw Components.results.NS_ERROR_NO_AGGREGATION;
    if (iid.equals(MYCOMPONENT_IID) ||
      iid.equals(Components.interfaces.nsISupports))
    return new nsExample();
```

```
    throw Components.results.NS_ERROR_INVALID_ARG;
  }
};

function NSGetModule(comMgr, fileSpec) { return nsExampleModule; }
```

8.1.4 C++ Beispiel Component

8.1.4.1 nsExampe.h

```
#ifndef __NSEXAMPLE_IMPL_H__
#define __NSEXAMPLE_IMPL_H__

#include "nsIExample.h"

#define MYEXAMPLE_CLASSNAME "nsExample2"
#define MYEXAMPLE_CONTRACTID "@mozilla.org/nsExample2;1"
#define MYEXAMPLE_CID_STRING {0x43F1C9C5, 0x258F, 0x4a68,{ 0x87, 0x29,
0xA2, 0x42, 0x5A, 0x1D, 0x66, 0x5A }}

static NS_DEFINE_CID(EXAMPLE_CID, EXAMPLE_CID_STRING);

class nsExample : public nsIExample
{
public:
  NS_DECL_ISUPPORTS
  NS_DECL_NSIEXAMPLE
  nsExample();
private:
  virtual ~nsExample();

};

#endif
```

8.1.4.2 nsExampe.cpp

```
#include "nsExample.h"

NS_IMPL_ISUPPORTS1(nsExample, nsIExample)

nsExample::nsExample() { }
nsExample::~nsExample() {}

NS_IMETHODIMP nsExample::AllCapital(const char *s, char **_retval)
{
  *_retval = static_cast<char*> (nsMemory::Clone(s,(strlen(s+1) *
sizeof(char)));
  for(int i = 0; i < s.length(); i++)
  {
    s[i] = toupper(s[i]);
  }
  return NS_OK;
}
```

8.1.4.3 Examplemodule.cpp

```
#include "nsIGenericFactory.h"

#include "nsExample.h"

NS_GENERIC_FACTORY_CONSTRUCTOR(nsExample)

static nsModuleComponentInfo components[] =
{
   {
     EXAMPLE_CLASSNAME,
     EXAMPLE_CID_STRING,
     EXAMPLE_CONTRACTID,
     nsExampleConstructor,
   }
};

NS_IMPL_NSGETMODULE(nsExampleModule, components)
```

8.1.5 jackftemplate.js (Seite 98)

```
const COMPONENT_IID = Components.interfaces.nsIClassInfo;
const COMPONENT_IID1 = Components.interfaces.nsIJackFExtension;
const COMPONENT_IID2 = Components.interfaces.nsIJackFNumericExtension;
const COMPONENT_IID3 = Components.interfaces.nsIJackFGraphicExtension;
const COMPONENT_IID4 = Components.interfaces.nsIJackFAlphanumericExtension;

const COMPONENT_CONTRACTID = '@mozilla.org/jackfjackftemplate;1';
const COMPONENT_CID = Components.ID('{500F8DF4-A41E-41f9-B22C-
  A063ED731C66}');
const COMPONENT_CLASSNAME = 'jackfjackftemplate';
const _logo = "chrome://jackftemplate/content/pics/icon.png";
const _version = "1.0";
const _name = "JackFTemplate";
const _description = "This is the template for all new Jack-F Pluggies.";

const _timeout = 30; // max timeconsumtion of the plugy in milliseconds
const _cachetime = 10; //the timeframe in seconds in which a valid result
  should be cached

function nsJackFExtension() { }

nsJackFExtension.prototype = {

  // nsIClassInfo
  contractID: COMPONENT_CONTRACTID,
  classDescription: COMPONENT_CLASSNAME,
  classID: COMPONENT_CID,
  implementationLanguage:
    Components.interfaces.nsIProgrammingLanguage.JAVASCRIPT,
  flags: Components.interfaces.nsIClassInfo.THREADSAFE,

  getInterfaces: function(count) {
    var ifaces = [
      COMPONENT_IID,
      COMPONENT_IID1,
      COMPONENT_IID2, //delete this line if your Extension doesn't support
        numeric results
```

```
        COMPONENT_IID3, //delete this line if your Extension doesn't support
          graphic results
        COMPONENT_IID4, //delete this line if your Extension doesn't support
          alphanumeric results
        Components.interfaces.nsISupports
      ];
      count.value = ifaces.length;
      return ifaces;
    },

    getHelperForLanguage: function(language) {
      return null;
    },

    QueryInterface: function(iid) {
      if (iid.equals(COMPONENT_IID) ||
        iid.equals(COMPONENT_IID1) ||
        iid.equals(COMPONENT_IID2) || //delete this line if your Extension
          doesn't support numeric results
        iid.equals(COMPONENT_IID3) || //delete this line if your Extension
          doesn't support graphic results
        iid.equals(COMPONENT_IID4) || //delete this line if your Extension
          doesn't support alphanumeric results
        iid.equals(Components.interfaces.nsISupports))
          return this;
      throw Components.results.NS_ERROR_NO_INTERFACE;
    },

  //P R O P E R T I E S   S T A R T
    get name() {
      return _name;
    },

    get description() {
      return _description;
    },

    get logo() {
      return _logo;
    },

    get version() {
      return _version;
    },

    get classID() {
      return COMPONENT_CID;
    },

  //sets the preferd time-out before an analysation will be terminated
    get timeout() {
      return _timeout;
    },

    get cachetime() {
      return _cachetime*1000;
  },
    //P R O P E R T I E S   E N D

  // has to be implemented if the Jack-F pluggy should be able to return an
  numeric result as a string
    analyseNumeric: function(msg) {
      ...
```

```
  },

//has to be implemented if the Jack-F pluggy should be able to return an
graphic result as a string
  analyseGraphic: function(msg) {
    ...
  },

//has to be implemented if the Jack-F pluggy should be able to return an
alphanumeric result as a string
  analyseAlphanumeric: function(msg) {
    ...
  },

//has to be implemented if the Jack-F pluggy should be able to convert an
numeric into a gaphic result
  convertToGraphic: function(score) {
    ...
  },

//has to be implemented if the Jack-F pluggy should be able to convert an
numeric into an alphanumeric result
  convertToAlphanumeric: function(score) {
    ...
  }
};

var nsJackFExtensionModule = {
  registerSelf: function(compMgr, fileSpec, location, type) {
    compMgr = compMgr.QueryInterface
      (Components.interfaces.nsIComponentRegistrar);
    compMgr.registerFactoryLocation(COMPONENT_CID,
                                    COMPONENT_CLASSNAME,
                                    COMPONENT_CONTRACTID,
                                    fileSpec,
                                    location,
                                    type);
    var catman = Components.classes["@mozilla.org/categorymanager;1"].
      getService(Components.interfaces.nsICategoryManager);
    catman.addCategoryEntry("JackF Pluggy", COMPONENT_CID,
      COMPONENT_CONTRACTID, true, true);
  },

  getClassObject: function(compMgr, cid, iid) {
    if (!cid.equals(COMPONENT_CID))
      throw Components.results.NS_ERROR_NO_INTERFACE;
    if (!iid.equals(Components.interfaces.nsIFactory))
      throw Components.results.NS_ERROR_NOT_IMPLEMENTED;
    return nsJackFExtensionFactory;
  },

  canUnload: function(compMgr) { return true; }
};

var nsJackFExtensionFactory = {
  createInstance: function(outer, iid) {
    if (outer != null)
      throw Components.results.NS_ERROR_NO_AGGREGATION;
    if (iid.equals(COMPONENT_IID) ||
        iid.equals(COMPONENT_IID1) ||
        iid.equals(COMPONENT_IID2) || //delete this line if your Extension
          doesn't support numeric results
        iid.equals(COMPONENT_IID3) || //delete this line if your Extension
```

```
            doesn't support graphic results
        iid.equals(COMPONENT_IID4) || //delete this line if your Extension
            doesn't support alphanumeric results
        iid.equals(Components.interfaces.nsISupports))
          return new nsJackFExtension();
      throw Components.results.NS_ERROR_INVALID_ARG;
  }
};

function NSGetModule(comMgr, fileSpec) { return nsJackFExtensionModule; }
```

8.2 Batch Datei für .xpi Paket

Das Original dieser Batch Datei ist von „Eric”[a]. Die unten vorliegende Datei wurde abgeändert damit der Ordner `components` in das Paket mit aufgenommen wird.

```
set x=%cd%
md build\chrome
md build\components
cd chrome
7z a -tzip "%x%.jar" * -r -mx=0
move "%x%.jar" ..\build\chrome
cd ..
copy install.rdf build
copy chrome.manifest build
cd build
copy ..\components\*.xpt .\components
copy ..\components\*.js .\components
7z a -tzip "%x%.xpi" * -r -mx=9
7z a -tzip "%x%.xpi" ../defaults -r -mx=9
move "%x%.xpi" ..\
cd ..
rd build /s/q
```

[a] http://roachfiend.com/archives/2004/12/08/how-to-create-firefox-extensions/

9 Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, am

10 Curriculum Vitae

Persönliche Daten

Name	Christian Haider
E-Mail	c_haider@aon.at
Geburtsdatum	24. September 1981
Geburtsort	Linz
Nationalität	Österreich
Familienstand	ledig
Hobbys	Golf Standard- und Lateintanz Squash

Ausbildung

Seit 10.2006	Johannes Kepler Universität Studium der Wirtschaftsinformatik Schwerpunkte: ▪ Information Engineering (strategische IT-Planung) ▪ Network Security ▪ Soziale Kompetenz (Konfliktmanagement, Rhetorik) ▪ Magisterarbeit: Design und Implementierung eines Frameworks zur Integration von sicherheitsrelevanten Erweiterungen in Mailclients
10.2005 – 05.2006	Sprachausbildung in Sydney, Australien EF Master Business English Course Abschluss: University of Cambridge Business English Certificate Higher Grade C
10.2002 – 09.2005	Johannes Kepler Universität Studium der Wirtschaftsinformatik
1996 – 2001	HTL für EDV & Organisation, Leonding Abschluss: Matura (Juni 2001)
1992 – 1996	Anton Bruckner Bundesrealgymnasium, Wels
1988 – 1992	Volksschule, Allhaming

Präsenzdienst

09.2001 – 05.2002	Jägerbataillon 15, Ebelsberg

Ferialpraktika

07.2000	Siemens AG, Wien Bereich: Banken (Programmierung)
07.1999	Siemens AG, Wien Bereich: Banken (Programmierung)
08.1998	Keba AG, Linz Bereich: Wareneingang
08.1997	Odörfer Ges.m.b.H., Linz Bereich: Kommissionierung

Berufspraxis

Seit 10.2006	BBRZ REHA Ges.m.b.H., Linz Freiberuflich Lehrender (Erwachsenenbildung) Schwerpunkte: ▪ PC-Hardware ▪ Netzwerke ▪ MS Client/Server
12.2003 – 09.2006	Marketing & Promotion Ges.m.b.H., Linz Angestellter ▪ Verantwortlich für die Wartung, Instandhaltung und Planung von 2 Windows 2003 Servern, 10 Windows XP Workstations, 4 MAC OSX Workstations, MS Client/Server, 3 VPN Verbindungen ▪ Neuverhandlung der 9 Standleitungen und 3 XDSL Anbindungen für Zweig- und Außenstellen
10.2006 – 05.2006	*Fernwartung aller Systeme aus Australien*
12.2003 – 03.2005	*Beschäftigt bei Bluestyle-Records Ges.m.b.H. (Linz), die im April 2005 mit der Marketing & Promotion Ges.m.b.H. (Linz) zusammengelegt wurde*
11.2005 – 12.2005	Leonardo Consulting, Sydney (Australien) Internship ▪ Erarbeitung der Vorteile von Aris für Softwareentwickler
07.2002 – 03.2003	BBRZ REHA Ges.m.b.H., Linz Freiberuflich Lehrender (Erwachsenenbildung) Schwerpunkte: ▪ PC-Hardware ▪ Netzwerke ▪ MS Client/Server
10.2000 – 08.2001	DLI-Datalink, Traun Freiberufliche Tätigkeit Schwerpunkte: ▪ Assemblieren und Testen von Computersystemen

Fremdsprachenkenntnisse

Englisch	sehr gut in Wort und Schrift