Universität Koblenz-Landau; Fachbereich 4: Informatik
AG Informationssysteme und Semantic Web (ISWeb)

Austausch strukturierter Informationen in Email über Wiki-Text

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von:
Markus Köhler
Mat.Nr.: 204210164
markusk@uni-koblenz.de

Beginn der Arbeit: 01. August 2008
Abgabe der Arbeit: 31. Oktober 2008

Betreuer Thomas Franz, AG Informationssysteme und Semantic Web (ISWeb)
Erstgutachter Prof. Dr. Steffen Staab, AG Informationssysteme und Semantic Web (ISWeb)

Koblenz, im Oktober 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Idee des Semantic Desktop hat in den letzten Jahren immer mehr an Aufschwung erfahren und ist mittlerweile ein intensives Forschungsgebiet. An der Universität Koblenz ist hierzu das Projekt X-COSIM[1] entstanden, welches im Wesentlichen aus den beiden Komponenten X-COSIMO (einer Ontologie) und X-COSIMA (der dazugehörigen API) besteht (FSA07). In diesem Kontext ist diese Studienarbeit der AG Staab (durch Prof. Dr. Steffen Staab und Thomas Franz) ausgeschrieben, in der eine Extension für einen Emailclient konzipiert und entwickelt werden soll, welche den Austausch strukturierter Daten (z.B. Termine, Aufgaben, Kontakte) über Email mithilfe einer (ebenfalls im Rahmen dieser Arbeit zu evaluierenden) Wiki-Markup-Language direkt im Body ermöglichen soll.
Zunächst möchte ich im Abschnitt Motivation die Vorzüge und den möglichen Nutzen der von mir zu entwickelnden Extension-Lösung aufzeigen. Im Anschluss daran findet sich die in Absprache mit den Betreuern festgelegte *Definition der Zielsetzung*. Hier sind die Mindestanforderungen an die abzugebende, fertige Extension festgehalten.
Im Abschnitt *Related Work* greife ich einen Fundus an Arbeiten auf, die sich um eine ähnliche Problemstellung oder generell mit Themen rund um verschiedene Aspekte dieser Arbeit (Markup-Languages, Semantic Desktop, Email-Prozesse, maschinelles Lernen) beschäftigen und mir im Vorfeld einen Überblick gegeben haben. Im Abschnitt *Definierte Wiki-Syntax* zeige ich anschließend meine Überlegungen zur Entwicklung einer Syntax und diskutiere Vor- und Nachteile verschiedener Entwürfe.
Im sechsten Kapitel wird genauer auf die technischen Aspekte der Realisierung der Extension eingegangen. Dieser Teil ist als Dokumentation zu Teilen des Quellcodes zu verstehen, um eine zukünftige Weiterentwicklung zu ermöglichen und (vor allem) zu erleichtern. Abschließend zu diesem Kapitel findet sich ein Usecase, welcher die zukünftige typische Verwendung der Extension verdeutlichen soll.
Am Ende der Arbeit gebe ich Ideen für eine mögliche Weiterentwicklung meiner Extension, ergänzt durch eine Reihe von Online-Ressourcen, die mir eine wertvolle Unterstützung bei der Extension-Entwicklung waren.

1 Cross(X)-COntext Semantic Information Management;
siehe http://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Research/x-cosim; Stand 25.10.2008

2 Motivation

Die Email etablierte sich bis heute als das Standard-Medium für weltweite Kommunikation und Interaktion in Firmen und Privathaushalten. Die einfache, kostenfreie Nutzung sowie enorme Verfügbarkeit sind mit Sicherheit weitere Faktoren, die die Ausweitung auf immer mehr Anwendungsfelder beschleunigen und begünstigen. Längst hat das Medium Email auch im mobilen Bereich Einzug gehalten (Blackberry) und nahezu jede Person im Internet nutzt eine Emailadresse und kann darüber kontaktiert werden (SHD08). Neben dem klassischen Einsatzfeld zur Kommunikation untereinander wird Email aber auch zum Versenden von Termininformationen, Kontakten und/oder (begünstigt durch immer größere Bandbreiten) auch Dateien verwendet. Dem gegenüber stehen eine Reihe von Applikationen (PIM, Dateimanager), die der Nutzer verwendet um die per Email versendeten Informationen später für sich zu speichern, weiter zu verwenden und zu verwalten.
Doch gerade diese verschiedenen Informationscontainer sorgen dafür, dass die gesammelten Informationen mit der Zeit immer weiter gestreut werden und ihren Zusammenhang verlieren. Einem gespeicherten Termin fehlt die Verbindung zu einer Email in deren Folge dieser Termin angelegt wurde, einem eingetragenen Kontakt sieht man nicht mehr an von welchem Kollegen man diese Informationen bekommen hat und auf wen man sich beispielsweise hätte berufen können. Es wäre also sehr praktisch einer Datei, einem Kontakt oder einem Termin auch nach dem Verschieben in seine endgültige Anwendung noch ansehen zu können, in welchem Kontext man diese Information erhalten hat. Wann und wer hat einem in welchem Zusammenhang diese Information zukommen lassen? An dieser Stelle greift X-COSIM ein und bietet eine Möglichkeit, kontextübergreifend semantische Informationen zu managen.
Darüber hinaus besteht in kommerziellen Lösungen wie Microsoft Outlook immer die Notwendigkeit zwischen verschiedenen Dialogen hin- und herzuspringen um zusätzliche Informationen, wie beispielsweise einen Termin, per Email zu versenden. Hierbei muss zunächst eine Email geschrieben und z.B. der entsprechende Termin dabei seperat angelegt und hinzugefügt werden. Sind unter den Empfängern der Email Benutzer die andere Emailclients verwenden so erhalten diese häufig nur unformatierte und unleserliche (da von Outlook automatisch generierte) Daten aus denen nur mit Mühe ein Termin abgelesen werden kann. Können die gesendeten Informationen zumindest von einer weiteren Applikation gelesen werden, so muss man dafür doch zumindest die Emailnachricht ansich verlassen um Informationen über die gesendeten Daten einsehen zu können (Hier kann es z.B. vorkommen, dass eine weitere Kalenderapplikation einen Termin aus einem speziellen Dateiformat, welches an die Email angehängt ist, auslesen kann). Gerade hier wäre ein Ansatz, der

es erlaubt, Informationen direkt in eine zu schreibende Email einzutragen/zu integrieren, welche dann gleichzeitig strukturiert und somit gut leserlich sind, eine komfortable Lösung. Ich möchte eine Extension entwickeln mit der ich es den Benutzern ermögliche, in Verbindung mit X-COSIM verschiedene Informationen schnell, strukturiert und unkompliziert einfach eingebunden in einen Emailtext auszutauschen und später (durch Einbindung in das X-COSIM Framework) leicht wiederzufinden. Mein Konzept sieht also zum Einen die Wahl einer Struktur vor, mit der die Benutzer die Informationen einfach in eine Email einbinden können. Zum Anderen sind die notwendigen Softwarekomponenten zu entwickeln die wiederum eine automatische Extraktion der Daten beim Empfänger ermöglichen.

3 Definition der Zielsetzung

In diesem Abschnitt der Arbeit sind die Anforderungen an die finale Software festgehalten. Das Ziel dieser Studienarbeit ist es allgemein, ein solches Szenario (wie unter dem Kapitel Motivation dargestellt) vereinfacht mit Mozilla Thunderbird als Open-Source PIM Anwendung[2] und dem Versenden von Kalendereinträgen, eingebettet in Wiki-Markup-Language im Body einer Email, zu realisieren.[3] Dabei soll auch die nahtlose Integration in X-COSIM beachtet werden. Das Plugin COSIMail[4] für Thunderbird ermöglicht es hierbei dem Emailclient, ein entsprechendes Framework (bestehend aus einer API X-COSIMA und einer Ontologie X-COSIMO (FSA07)) zu verwenden. Um auch meine Extension an X-COSIM anbinden zu können ist eine Aufspaltung von COSIMail geplant. Hierbei soll eine API zum Abspeichern der semantischen Informationen aus Thunderbird heraus von COSIMail gelöst werden, um in meiner (und möglichen weiteren Extensions) Verwendung finden zu können.[5]
Die folgenden konkreten Anforderungen sind in beidseitiger Absprache mit dem Betreuer als Mindestanforderung an die im Rahmen dieser Studienarbeit zu entwickelnde Extension festgehalten.

Technische Anforderungen:

- Jede eingehende Email wird beim Öffnen automatisch nach Informationen zu Terminen/ Kalendereinträgen durchsucht
- Die Informationen werden zusätzlich in X-COSIM (via einer API) weggeschrieben[6]
- Die Struktur muss dabei so gewählt sein, dass eine einfache Erweiterung auf weitere Informationspakete (wie z.B. Aufgaben) möglich ist

Anforderungen an das User-Interface:

- Wird eine Information erkannt, wird der Benutzer darüber automatisch (visuell) informiert
- Die Extension ruft dann auf Wunsch des Benutzers die Funktion zum Anlegen eines neuen Termins auf und übergibt die extrahierten Daten aus der Email an die entsprechenden Felder

2 Thunderbird ist für alle gängigen Plattformen verfügbar. Damit erreichen wir eine plattformunabhängige Funktion, da auch die Extension in jedem beliebigen Betriebssystem innerhalb von Thunderbird eingesetzt werden kann.
3 Natürlich gibt es bereits mit Microsoft Outlook kommerzielle Lösungen, die das Versenden von Terminen per Email ermöglichen. In diesem Fall ist aber ein spezielles Dateiformat nötig, über das die Informationen übermittelt werden.
4 siehe http://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Research/cosimail; Stand 10.09.2008
5 Dies ist jedoch nicht Teil dieser Studienarbeit und wird seperat von David Mann (dem Entwickler von COSIMail) umgesetzt.
6 Leider war zum Zeitpunkt der Abgabe der Studienarbeit diese API nicht fertiggestellt. Nach Absprache mit dem Betreuer bleibt dieser Punkt damit außen vor.

4 Related Work

In diesem Abschnitt möchte ich Paper und Artikel hervorheben, die den Nutzen einer solchen Extension/einer solchen Funktionalität bereits analysiert und erkannt haben. Außerdem sollen ähnliche Lösungsansätze vorgestellt werden um einen breiten Überblick über die Thematik zu bekommen.
So argumentiert das Paper *Semantic Email as a Communication Medium for the Social Semantic Desktop* (SHD08), dass gerade wegen der Einfachheit und der weiten Verbreitung der Email als Medium beim semantischen Desktop besonders viel Bedeutung zugesprochen werden muss. Das Paper zeigt eine Lösung auf, bei der aus einer Emailkonversation heraus eine Aufgabenverwaltung generiert und gesteuert wird. Dem Verfahren liegt dabei ein Workflow-Modell zugrunde. Emails werden hierbei also nicht mehr nur zur Übertragung von einfachen Texten verwendet, sondern dienen den Benutzern als Medium zum Übermitteln von strukturierten Daten.
Ein weiteres Anwendungsfeld ist das Programm *Jourknow*[7], welches versucht, sowohl eine einfache Eingabe für den Benutzer als auch eine ausreichende Aufnahme von Zusatzinformation zur automatischen Einordnung und späteren Suchbarkeit notierter Informationen zu realisieren. Entweder geht der Benutzer davon aus viele der Kontextinformationen später nicht zu gebrauchen oder aber es wäre ohnehin zu viel Aufwand diese alle festzuhalten. Um den Benutzer bei seiner normalen Arbeit zu unterstützen ermöglicht das Programm also zum Einen die einfache Eingabe von Textinformationen, erweitert werden diese simplen Eingabemechanismen jedoch von einer automatischen Kontext-Aufnahme (so werden z.B. Informationen über zum Zeitpunkt der Aufnahme besuchte Websites gespeichert, es werden Screenshots angefertigt und hinzugefügt, über eine Webcam wird die Umgebung als Foto zur Information hinzugefügt, der Ort der Aufnahme geokodiert gespeichert und vieles mehr). Denn, so argumentiert das Paper, gerade der Kontext ermöglicht es dem Benutzer später seine Notizen wieder einzuordnen und Verbindungen herzustellen. Innerhalb der freien Texteingabe werden dem Benutzer zudem eine Reihe an Keywords mitgegeben (eine sog. pidgin-Grammatik), mit denen er seinen Text zusätzlich schnell und unkompliziert strukturieren kann. Beispielsweise kann er eine Notiz im Handumdrehen durch das Keyword 'todo' in eine Aufgabe umwandeln.
Weitere Paper wie (KLDK06) argumentieren, dass sich Email inzwischen als die Anwendung schlechthin zur Kommunikation und zum Managen von Aufgaben und komplexeren Arbeiten durchgesetzt hat. Gleichzeitig kristallisiert dieses Paper aber auch die Problematik heraus, die sich

7 siehe http://projects.csail.mit.edu/jourknow/; Stand 11.09.2008

automatisch ergibt da die Email-Anwendung ansich kein Wissen über den Kontext einer jeweiligen Email hat. Es liegt damit in der Hand des Benutzers seine Aktivitäten wie z.B. Online-Einkäufe selber zu managen, er muss selber den Überblick bewahren und den Status tracken. Das Paper zeigt auf, welche Lösungsansätze und Anwendungen für *Activity-Centric-Email* bereits existieren und welcher Gewinn sich daraus für die Anwender ergibt. Dabei werden grundsätzlich zwei Arten von Aktivitäten unterschieden: starr strukturierte Aktivitäten (wie oben bereits genannt am Beispiel eines Online-Einkaufs) sowie unstrukturierte dialogorientierte Aktivitäten (wie die Organisation von Meetings). Um eine klar strukturierte Aktivität darzustellen wird hier (anders als bei (SHD08)) ein endlicher Automat verwendet, bei dem Email-Nachrichten die Übergänge zwischen verschiedenen Stati darstellen. Zudem wird der Status einer Aktivität in verschiedenen Darstellungen dem Benutzer in seinem Email-Client zur Verfügung gestellt. Der Prozess einer Aktivität besteht aus den Schritten der Erkennung und der jeweiligen Identifikation der Übergänge.
Wie wichtig eine einfache Bedienung und ein leichter Einstieg für die Akzeptanz einer neuen Technik/ Methode ist zeigt sich in (MEH04). Im Fokus stehen dabei *semantische Email-Prozesse*[8], welche bisher manuell durchgeführte Email-basierte Aktivitäten automatisieren und somit Zeit sparen und Fehler vermeiden. Die Autoren spezifizieren eine Reihe von Vorlagen, welche (einmal von einer geübten Person erstellt) beliebig oft instantiiert und verwendet werden können. Ganz konkret wird dies am Beispiel der Planung eines abgestimmten amerikanischen Potluck[9] dargestellt. Der Organisator kann somit eine Vorlage (eines Autors) instantiieren und über ein Webformular (welches über einen Form-Generator mit Parametern und Beschreibungen generiert wird) mit Daten füllen. Dennoch muss das Team am Ende das Fazit ziehen, dass selbst die Verwendung von Vorlagen immer noch nicht ausreicht um eine ausreichende Akzeptanz für die Verwendung von SEPs zu schaffen. Zusätzlich wird das Verifizieren der Templates im Paper behandelt. Eine Verifikation ist notwendig, um einen reibungslosen Ablauf auf Basis der verwendeten Templates zu gewährleisten.[10]

8 semantic email processes, SEP
9 Ein Treffen, bei dem jeder Gast ein oder mehrere Speisen mitbringt. So entsteht schnell eine Art Buffet.
10 Darüber hinaus ist generell auch die Verifikation der gefundenen Datensätze sinnvoll. Wie und in welchem Umfang dies durchzuführen ist zeigt das Paper *Evaluating Machine Learning for Information Extraction* (ICC$^+$05) auf.

5 Definierte Wiki-Syntax

Um die relevanten Blöcke (und damit die relevante Information) innerhalb eines Email-Body zu erkennen prüft die Extension den Text auf bekannte Keywords, die (je nach Typ) die entsprechenden Informationen im Body kennzeichnen.
Hier war mein Ziel auf der einen Seite, dass die Informationen trotz der Markup-Language vom Benutzer einfach aus der Email heraus gelesen und (natürlich) auch erstellt werden können, auf der anderen Seite aber müssen sie für die Extension eindeutig genug sein, um die Daten korrekt zu erkennen. Ein erster Gedanke war (stark angelehnt an die Wiki-Markup-Language[11]) folgende Struktur:
Für die Unterscheidung der verschiedenen Elemente war folgende Bezeichnung gedacht:

- 'Cal' für Termin
- 'Task' für Aufgabe
- 'Contact' für Kontakt

Der nachfolgende Ausdruck sollte dann dem Beginn des Termins entsprechen und folgendermaßen (am Beispiel eines Kalendereintrags) erweitert werden:

<Cal>	Beginn der Termininformation
<CalTitle>	Titel
<CalDateStart>, <CalDateEnd>	Datum
<CalTimeStart>, <CalTimeEnd>	Uhrzeit
<CalLoc>	Ort
<CalNote>	Notiz
</Cal>	Ende der Termininformationen

Ein Eintrag würde dann folgendermaßen aussehen: [[Cal:Mittagessen(<CalDateStart>15.04.2008 <CalTimeStart>14.00< CalDateEnd>15.04.2008<CalTimeStart>16.00)]]

Allerdings ist diese Darstellung (ohne Zweifel) vom Benutzer nur noch schwer zu lesen bzw. kaum zu erstellen, zu viele Klammerungen müssen beachtet werden. Auch sorgt die Unterscheidung von Datum und Uhrzeit für weitere Keywords. Um dieser Problematik entgegenzuwirken orientierte sich mein zweiter Ansatz an dem Entwurf der Kombination von Tags und den so genannten 'pidgin

11 siehe http://en.wikipedia.org/wiki/How_to_edit; Stand 28.10.2008

grammars' (KBKS07). Demnach soll folgende Markup-Language verwendet werden:

@Termin	Titelinformation eines Termins
@Start	Startzeit im standardisierten Format MM/DD/YYYY HH:MM
@Ende	Endzeit im standardisierten Format MM/DD/YYYY HH:MM
@Notiz	Notiz
@Ort	Ort

Der zuvor verwendete Beispieleintrag würde mit dieser Markup-Language dann folgendermaßen aussehen (und könnte z.B. einfach um den Ort erweitert werden):

@Termin Mittagessen
@Start 04-15-2008 14-00
@Ende 04-15-2008 16-00
@Ort D-Gebäude

Der Vorteil ist direkt zu erkennen: Die Regeln sind sehr einfach (ein '@' gefolgt von einem Namen, der ein Feld des jeweiligen Informationselements beschreibt) und können somit beliebig erweitert und auf andere Objekte übertragen werden. Durch die Zeilenumbrüche (als Begrenzung) nach jeder Information kann auch ohne einen speziellen Parser die Information bereits vor der Erkennung und Speicherung durch die Extension vom Benutzer erkannt und verstanden werden.
Zudem kann die Information somit auch nahtlos in weiteren Text eingebettet werden und somit auf natürlichsprachlichem Weg gelesen und verstanden werden, ist gleichzeitig aber auch eindeutig genug für die Verarbeitung durch die Extension. Nachfolgend ein Beispiel:

Hallo Michael,
wie wäre es mit dem @Termin Mittagessen
von @Start 04-15-2008 14-00
bis @Ende 04-15-2008 16-00
?
Wir könnten uns im @Ort D-Gebäude
treffen!

6 Realisierung

6.1 Technische Basis zur Implementierung

Für die Implementierung des Extension habe ich folgende virtuelle Entwicklungsumgebung konfiguriert:

- Ubuntu Linux 8.04.1 Desktop Edition

  Java 1.6.0

  Thunderbird 2.0.0.14

    Lightning 0.8

    DOM Inspector

    COSIMail 0.3.0.2

  Apache Tomcat Webserver 6.0.16

    LENA 1.0.2

Im Hintergrund läuft ein Apache-Tomcat-Webserver[12] mit LENA[13], einem an der Universiät Koblenz entwickelten Tool zur Darstellung von RDF-Daten. Dies soll dazu dienen, die im nativen Repository von COSIMail weggeschriebenen Daten visualisieren und (für den Entwickler) überprüfen zu können.
Da Thunderbird in der momentan aktuellen Version 2.0.0.14 keine Termin- und Aufgabenverwaltung unterstützt wird hierfür die so genannte Lightning-Erweiterung[14] in der Version 0.8 verwendet. Programmiert wird die Extension in Javascript.

6.2 Features

Neben der im Abschnitt *Definition der Zielsetzung* formulierten Funktionen habe ich folgende Features/Unterstützungen in meine Extension implementiert und eingebunden:

12 siehe http://tomcat.apache.org/; Stand 11.09.2008
13 siehe http://isweb.uni-koblenz.de/Research/lena; Stand 11.09.2008
14 siehe https://addons.mozilla.org/de/thunderbird/addon/2313; Stand 22.10.2008

- Multilingualität: Die Extension ist multilingual ausgelegt. Sämtliche Beschriftungen in Dialogen werden aus so genannten dtd-Dateien bezogen. Im Source-Folder der Extensions befindet sich der Ordner *locale* (hier wiederum der Ordner *de-DE* und *en-EN*), der die jeweiligen Übersetzungen enthält.[15]
- Flexible Keywords: Der Benutzer hat die Möglichkeit, die Standard-Keywords (Termin, Start, Ende, Ort, Notiz) durch selbst gewählte Keywords zu ersetzen. Dazu bietet die Extension ein Einstellungsmenü, in welchem die Keywords einfach eingetragen werden können (siehe Abbildung 6.1).[16]

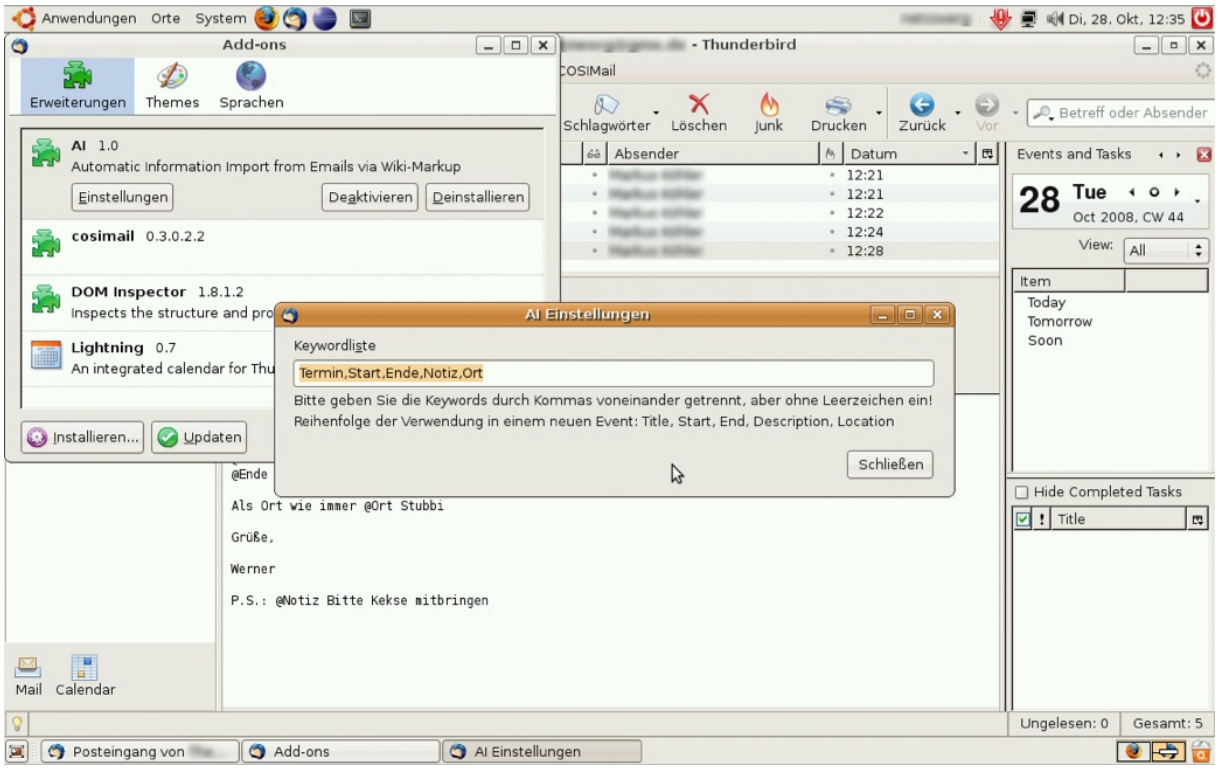

Abbildung 6.1: AI Einstellungen mit den Standard-Keywords.

15 Weitere Informationen zur Multilingualität von Thunderbird-Extensions, sowie deren genaue Funktionalität finden sich unter anderem auf http://www.xulplanet.com/tutorials/xultu/locale.html; Stand 26.10.2008

16 Zu beachten sind hier aber die Einschränkungen bezüglich der Reihenfolge. Dazu mehr in Kapitel *7.1 Einschränkungen*.

6.3 Code-Dokumentation

In diesem Abschnitt werden ausgewählte Codezeilen vorgestellt, die die Vorgehensweise meiner Extension verdeutlichen sollen.
Die Hauptfunktionalität von *AI* ist in der *overlay.js*-Datei im Ordner *content* des Source-Folder integriert. Nach dem Start werden zunächst die Keywords des Users aus dem Einstellungen-Dialog geladen. Dazu wird ein Zweig innerhalb der DOM-Hierarchie von Thunderbird aufgebaut der bis zum Einstellungsdialog meiner Extension reicht (Zeile 4). Die Werte selber befinden sich dann als String im Objekt *extensions.ai.stringpref*. Um die Werte abschließend als Felder in einem Array zu haben wird der String an den Kommastellen aufgeteilt (Zeile 8).

Listing 6.1: Laden der Keywords

```
1  //This function returns an array containing the keywords specified by the user in the preferences dialog
2  function getkeywordprefs(){
3    //Create a preference branch from the top-level
4    var prefs = Components.classes["@mozilla.org/preferences-service;1"].getService(Components.interfaces.nsIPrefBranch);
5    //Get the parameters (e.g. Termin, Start, Ende...) as a string
6    var keywordsstring = prefs.getCharPref("extensions.ai.stringpref");
7    //Convert this string into an array
8    var keywordarray = keywordsstring.split(",");
9    return keywordarray;
10 };
```

Das ausgelesene Array enthält somit alle Keywords nach denen später in den Emails gesucht werden soll und die entsprechend die relevanten Informationen kennzeichnen. Um relevante Informationen erkennen zu können muss zunächst noch der zur momentan vom Benutzer ausgewählten Email zugehörige Body ermittelt werden. Dies funktioniert über eine Funktion (*getemailbody*), die bereits von David Mann im Kontext von COSIMail umgesetzt wurde. Um in meiner Extension verwendet werden zu können waren weitere kleinere Anpassungen notwendig. Der Body liegt anschließend in Form eines String vor, nun können daraus die relevanten Daten ermittelt und in einem Array zurückgegeben werden. Dazu werden sowohl der Body als auch die Keyword-Liste als Parameter an die Funktion *extractInformation* übergeben. Diese sucht zunächst nach dem Keyword-Indikator @ (Zeile 14) und überprüft dann, ob dahinter ein gültiges (also vom Benutzer definiertes) Keyword steht (*nextkeywordis*, Zeile 19). Ist dies der Fall, muss der darauf folgende Text bis zum nächsten Zeilenumbruch (Zeile 28) extrahiert werden, da es sich dabei um die relevante Information handelt (Die String-Funktion *slice* schneidet hierbei den entsprechenden Abschnitt heraus). Über die Switch-Anweisung am Ende (Zeile 32) wird dann noch das zu dem gefundenen Keyword zugehörige Feld des Rückgabe-Array gesucht und die Information entsprechend eingetragen. Anschließend untersucht die Routine den restlichen Teil des Body auf weitere Keywords.

Listing 6.2: Extraktion der Daten (Auszug)

```
11 function extractInformation(emailbody, keywordarray){
12   ...
13   //Find the position of the next keywordindicator @
14   pos = emailbody.indexOf("@");
15    ...
16     //Throw away everything before the first keywordindicator and also the indicator
17     emailbody = emailbody.slice(pos+1);
```

```
18      //Find out which keyword comes next
19      nextone = nextkeywordis(emailbody, keywordarray);
20         ...
21        //Now u know the keyword, so remove it from the emailbody
22        //e.g. remove "Termin ", Termin+blank
23        emailbody = emailbody.slice((nextone.length+1));
24        /*Next step: get the data belonging to the detected keyword
25        this will be the next part (which now is at the beginning of the emailbody)
26        we assume that the data is ended by a \n-Newline */
27        //So get the next "newline"
28        pos = emailbody.indexOf("\n");
29        /*...and now the data is emailbody.slice(0,pos)
30        So find out the position in the detectedinfo-array to store the data depending
31        on the keyword which was found, this is the default order */
32        switch (nextone) {
33          case keywordarray[0]: //will be filled into "Title"
34            detectedinfo[0]=emailbody.slice(0,pos);
35            break;
36            ...
37    //return the array with all the detected information
38    return detectedinfo;
39  };
```

Übergibt man an diese Funktion entsprechend andere Keywords (wie z.B. *Titel* anstelle von *Termin*), so erhält man dann auch nur die Informationen zurück die hinter diesen Keywords stehen. Die Funktion gibt ein Array zurück (Zeile 38), welches die gefundenen Informationen (als Strings) in einer vordefinierten Reihenfolge enthält. Durch die Funktion *nextkeywordis* wird verhindert, dass einfache Emailadressen als zu extrahierende Informationen erkannt werden. Für den Fall dass kein gültiges Keyword gefunden wird gibt die Funktion dabei einfach *none* zurück und verhindert somit eine Extraktion von Daten.

Listing 6.3: Ermitteln des folgenden Keywords

```
40  function nextkeywordis(emailbody, keywordarray){
41    numofkeywords = keywordarray.length;
42    var nextkeyword = 0;
43    var i = 0;
44    //Check what keyword is in front of the emailbody
45    //while no keyword was found yet and you have not looked for all keywords...
46    while (nextkeyword == 0 && i < numofkeywords){
47      //=0 means keyword was found at the beginning directly after the @
48      if(emailbody.indexOf(keywordarray[i])==0)
49        //Then return the keyword
50        return keywordarray[i];
51      //Otherwise go on w/ the search
52      i++;
53    }
54    return "None";
55  };
```

Die erkannten Informationen können anschließend direkt weiterverwendet werden. Eine Ausnahme stellen die gefundenen Zeit-Informationen dar. Diese müssen noch über die Funktion *getcorrectDate-Format* in das so genannte calIDateTime-Format umgeformt werden.
Abschließend wird der New Event-Dialog der Lightning-Erweiterung aufgerufen (Zeile 69). Hierbei werden Parameter und Eigenschaften der neuen Termin-Instanz mit den zuvor gefundenen Daten belegt (Zeile 61-67).

Listing 6.4: Übergabe der gefundenen Daten (Auszug)

```
56      const CI = Components.interfaces;
57      const CALMGR_CONTRACTID = "@mozilla.org/calendar/manager, 1";
```

```
58      //create a new event
59      var newevent = Components.classes["@mozilla.org/calendar/event;1"].createInstance(CI.calIEvent);
60      //Set the title w/ the informationen extracted from the emailbody...
61      newevent.title = detectedinformation_array[0];
62      //..and all the other information
63      newevent.setProperty("LOCATION", detectedinformation_array[4]);
64      newevent.setProperty("DESCRIPTION", detectedinformation_array[3]);
65      //Special treatment: Time
66      newevent.startDate = getcorrectDateFormat(detectedinformation_array[1]);
67      newevent.endDate = getcorrectDateFormat(detectedinformation_array[2]);
68      //Open the dialog and show the information
69      createEventWithDialog(null, null, null, null, newevent);
```

In der momentanen Version werden (wie in Listing 6.4 gezeigt) die zuvor gefundenen Daten in einer festen Reihenfolge an einen neuen Termin übergeben. Um die hier gezeigte prototypische Anwendung auf andere Felder (wie z.B. Aufgaben oder Kontake) zu erweitern müssen dann eine Unterscheidung der verschiedenen Informationstypen sowie weitere Keywords eingeführt werden. Wenn die Extension die Art der Information in der Email ermittelt hat können die Daten dann an weitere spezifische Felder eben dieses Datentyps[17] übergeben werden.

Über einen Event-Listener werden die Funktionen zum Auslesen des Email-Body sowie dem Untersuchen nach Keywords (und damit verbundenen Daten) immer dann ausgeführt, wenn der Benutzer eine Email anklickt [18]. Dadurch kann die Extension sofort entscheiden, ob eine Email überhaupt relevante Daten enthält.

Listing 6.5: Event-Listener

```
70  //Add an event-listener observing if a user clicks on a email
71  document.getElementById("threadPaneBox").addEventListener("click", informuser, true);
```

Ist dies der Fall[19], wird der entsprechenden Email das Schlüsselwort *AI:Informationen_gefunden* [20] zugewiesen (Zeile 76) und der Benutzer somit darüber informiert. Thunderbird ordnet jeder Email eine String-Eigenschaft mit dem Namen *keywords* zu. Diese kann entsprechend ausgelesen und um ein eigenes Schlüsselwort erweitert werden.

Listing 6.6: Hinzufügen eines Schlüsselworts (Auszug)

```
72  var taglist = selectedmessage.getStringProperty("keywords");
73  //check if there are already some tags in this email
74  if (taglist.split(" ").indexOf(AI_TAG.toLowerCase()) < 0){
75    taglist += " ";
76    taglist += AI_TAG.toLowerCase();
77    //because otherwise the setStringProperty will override all previously set tags!
78    selectedmessage.setStringProperty("keywords", taglist);
79  };
```

Über die hier aufgeführten Erklärungen hinweg ist der gesamte Quelltext ausführlich dokumentiert, um dem Programmierer bei der Erweiterung meiner Extension (um z.B. das Erkennen von Aufgaben oder Kontaktdaten) zu unterstützen.

17 Mit Datentyp ist in diesem Zusammenhang ein Termin, eine Aufgabe oder ein Kontakt gemeint.
18 Diese Aktionen werden dann in der Funktion *informuser* aufgerufen.
19 Momentan informiert die Extension der Benutzer wenn mindestens ein Keyword gefunden wurde.
20 Der Unterstrich ist notwendig, da die Schlüsselwörter im System durch Leerzeichen voneinander getrennt werden. Andernfalls würde das Schlüsselwort demnach in mehrere Schlüsselwörter zertrennt.

Hinweis: Der gesamte Quelltext meiner Extension sowie eine kompilierte .xpi-Version (zur direkten Installation) befindet sich im SVN-Repository der AG ISWeb im Ordner *markusk*. Zugang erhält man ggf. über Thomas Franz.

Im folgenden Abschnitt wird ein Usecase beschrieben, der den Ablauf des Informationsaustauschs via Email in Thunderbird und das anschließende Erkennen und Weiterverarbeiten von Informationen durch meine Extension verdeutlichen soll.

6.4 Usecase

Dieser Abschnitt soll einen typischen Anwendungsfall skizzieren. Dabei bekommt ein Benutzer per Email Termininformationen zugesendet, welche in die (speziell für diese Studienarbeit entwickelte) Markup-Language eingebettet sind und von der Extension erkannt werden, um den Austausch strukturierter Informationen via Email zu ermöglichen. Die Extension erkennt Teile des Bodys einer eingehenden Email und verwendet diese, um ein entsprechendes Objekt (im konkreten Fall ein Termin) in Thunderbird anzulegen. Diese können vom Benutzer dann in ganz normalem Umfang (wie auch selbst angelegte Objekte) weiterverwendet werden.

6.4.1 Schritt 1

Der Benutzer erhält eine Email und öffnet diese durch anklicken.

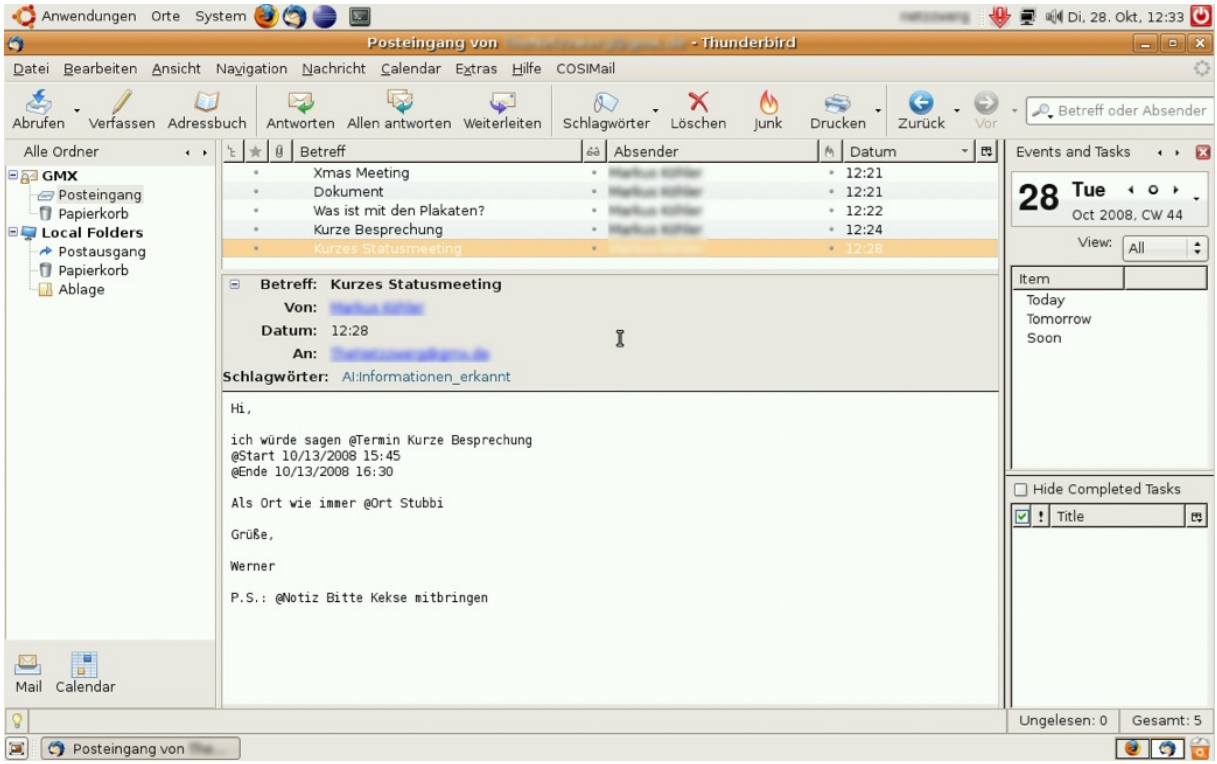

Abbildung 6.2: Schritt 1: Der Benutzer empfängt eine Email und liest diese.

Das Schlüsselwort *AI:Informationen_erkannt* im Header der Email weist ihn darauf hin, dass die Extension Informationen in dieser Email erkannt und verarbeitet hat.

6.4.2 Schritt 2

Da der Benutzer die erkannten Informationen speichern möchte klickt er mit der rechten Maustaste auf die Email. Im sich daraufhin öffnenden Kontextmenü (siehe Abbildung 6.3) wählt er die Option *AI: Erkenne Termininformation*.

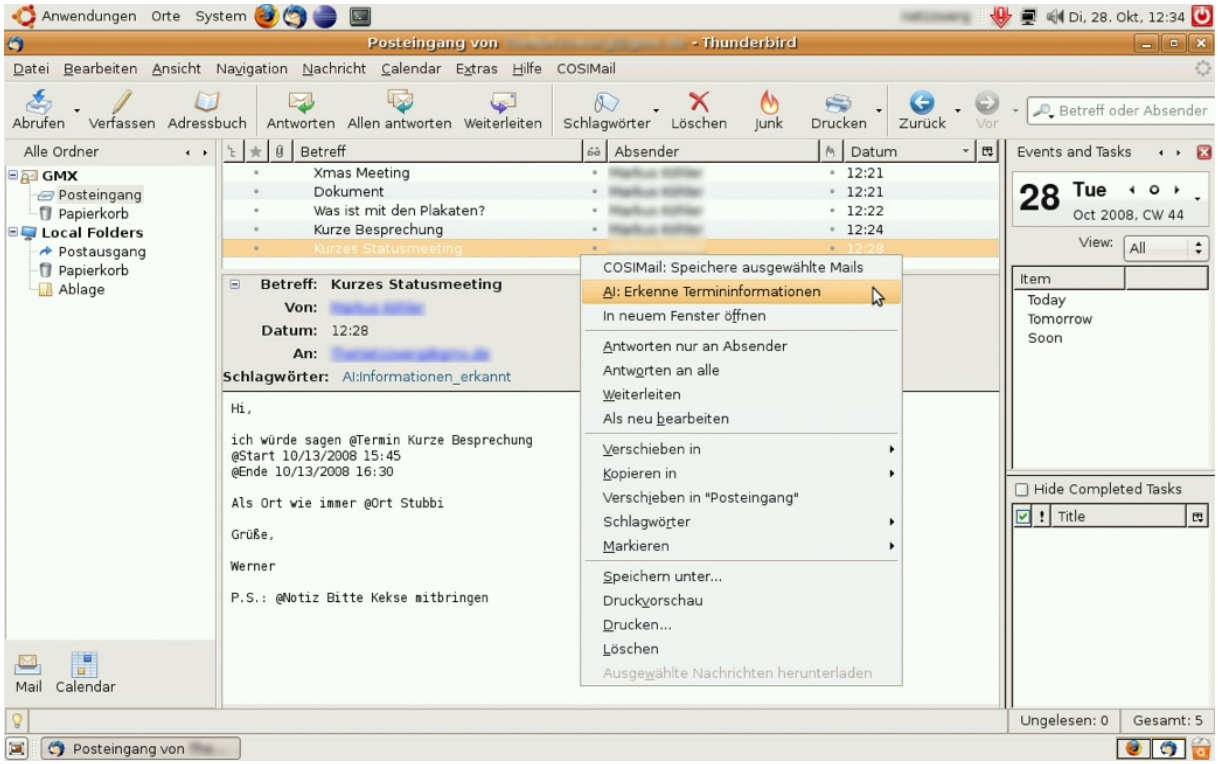

Abbildung 6.3: Schritt 2: Der Benutzer wählt die Option zum Speichern der Termininformation.

6.4.3 Schritt 3

Es öffnet sich der Dialog der Lightning-Erweiterung von Thunderbird um einen neuen Kalendereintrag anzulegen. Hierbei sind die Felder *Title*, *Location*, *Start*, *End* und *Description* bereits mit den von der Extension erkannten Werten gefüllt.

Eventuell fügt der Benutzer zusätzliche Informationen zum Termin hinzu (z.B. eine Erinnerung). Durch einen Klick auf *Save and Close* speichert sich der Benutzer den Kalendereintrag anschließend ab. Abbildung 6.5 zeigt den daraus generierten Eintrag im Kalender.

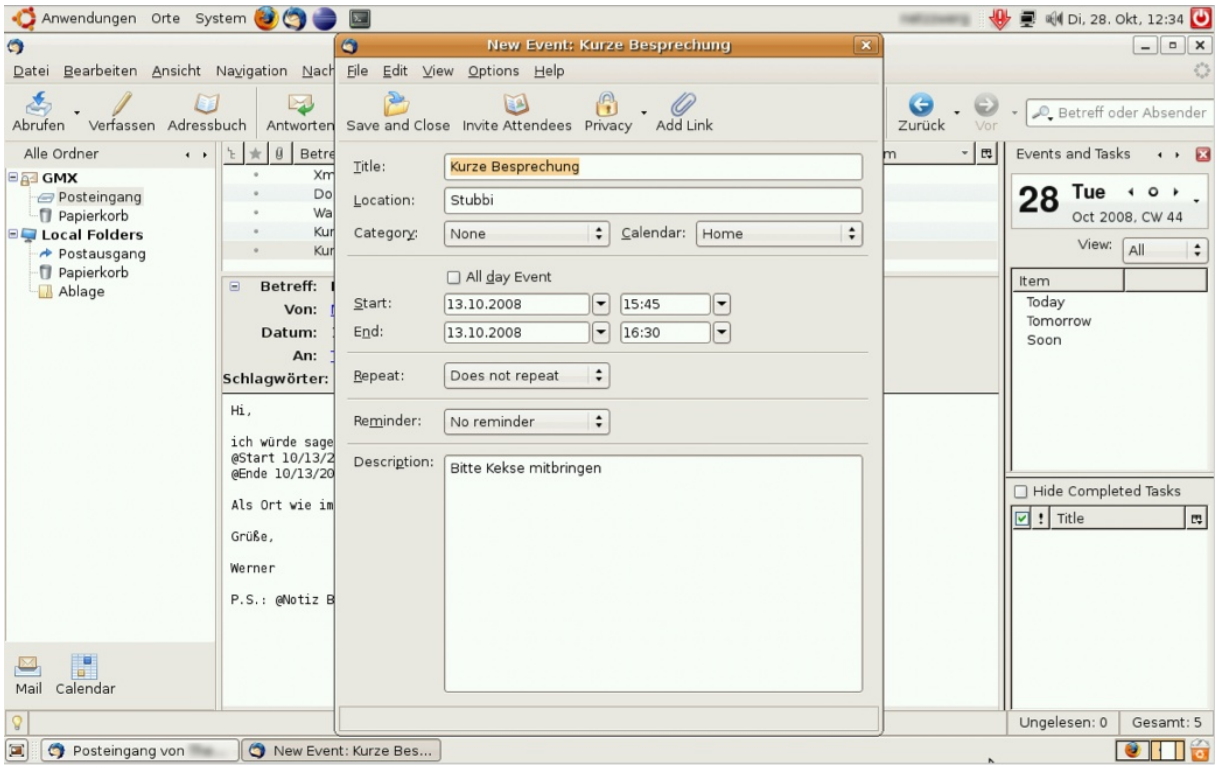

Abbildung 6.4: Schritt 3: Der Benutzer überprüft die erkannten Daten.

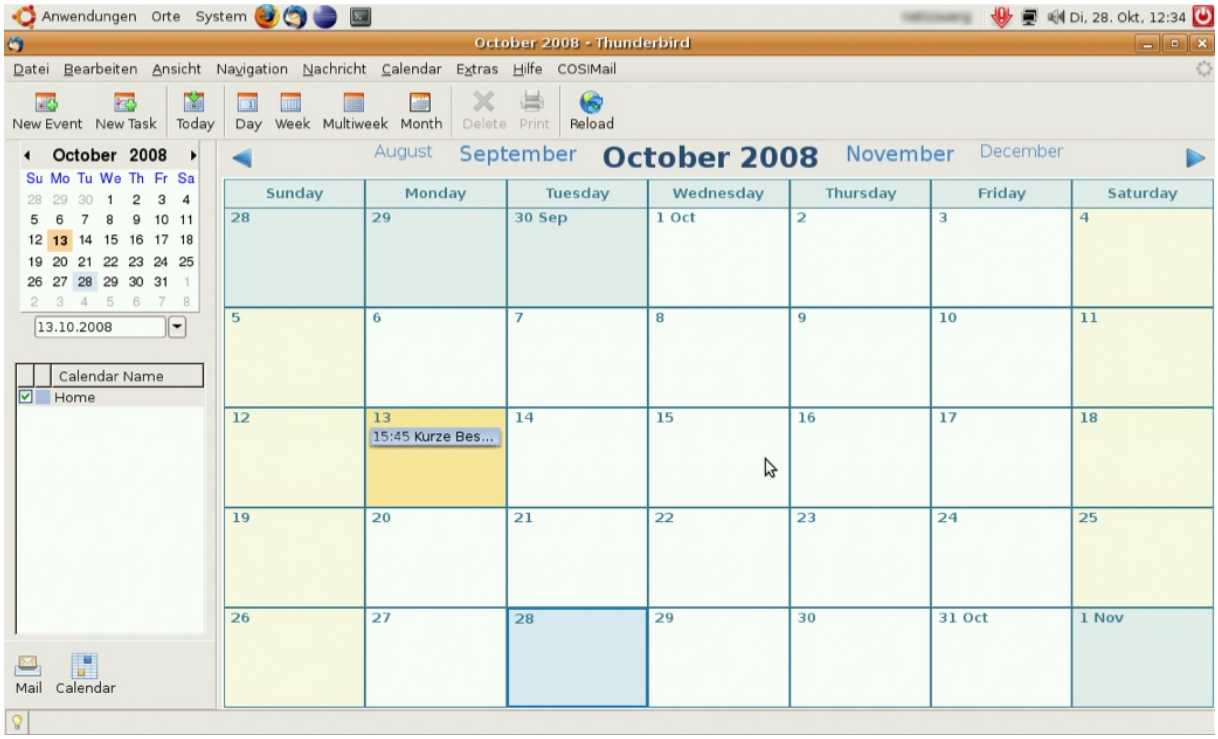

Abbildung 6.5: Der erstelle Eintrag im Kalender.

7 Ausblick

7.1 Einschränkung

Aufgrund technischer Aspekte/Probleme ist es momentan nur möglich Daten aus Emails von POP3-Servern oder offline geschalteten IMAP-Postfächern zu extrahieren. Bei Emails die auf (online) IMAP-Servern liegen ist dies z.Z. nicht möglich. Die zuvor beschriebene Funktion um den Body einer Email auslesen zu können fängt diese Option ab und blendet ggf. ein Pop-Up mit entsprechenden Informationen für den Benutzer ein.
Bei der Eingabe der Start- und Endzeit muss sich der Benutzer streng an das Format MM/DD/YYYY HH:MM halten, um die automatische Umwandlung in das Lightning-spezifische Kalenderformat zu ermöglichen. Bei entsprechend abgewandelter Eingabe erkennt die Extension die Daten ggf. nur fehlerhaft.
Diese beiden Aspekte könnten in einer Weiterentwicklung verbessert werden. Die Funktionalität für online IMAP-Postfächer ist in COSIMail realisiert worden, lies sich im Kontext meiner Extension aber nicht umsetzen. Hinweise hierzu finden sich im Quelltext von COSIMail, entsprechende Ansätze sind in meiner Extension durch Kommentare gekennzeichnet.

7.2 Weitere Funktionen

Wie bereits zur Einleitung erwähnt kann und soll meine Extension um weitere Anwendungsfelder erweitert werden. Eine zukünftige Erkennung von Aufgaben und Kontaktdaten ist hier denkbar. Dabei muss ein Weg gefunden werden, auch verschiedene Informationspakete innerhalb einer Email zu managen.
Meine Vision ist dabei ein dynamischer/interaktiver Einstellungsdialog für AI. Hier kann der Benutzer wieder selber seine Keywords festlegen, die daran geknüpfte Information aber graphisch an die Felder der spezifischen Dialoge in Lightning anbinden. Was damit gemeint ist soll folgende Abbildung verdeutlichen:

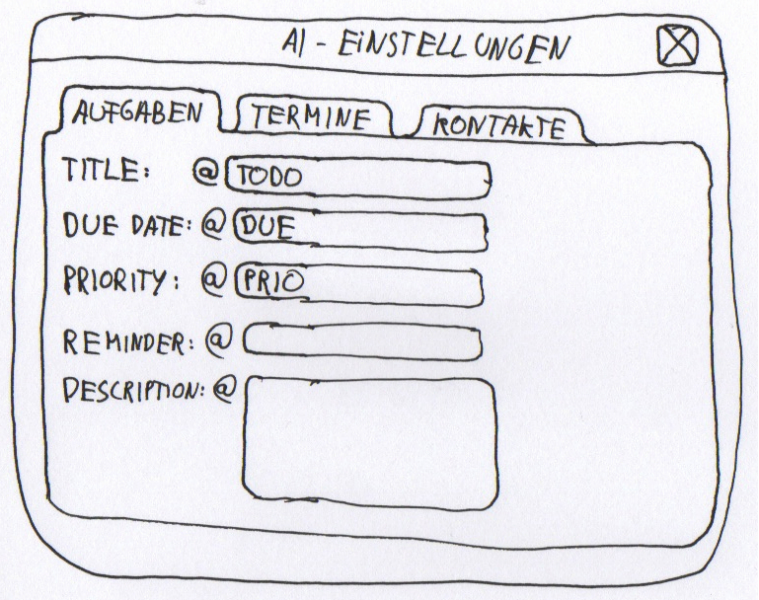

Abbildung 7.1: Mock-Up eines interaktiven Einstellungsdialog

Der Benutzer sieht alle Felder die das jeweilige Objekt in Lightning beinhaltet (hier z.B. die Aufgaben). Nun kann er seine gewünschten Keywords einfach in die Textboxen dahinter eintragen. Die Extension generiert die Zusammenhänge automatisch und erkennt Informationen zum einen durch die Definition des Benutzers, zum Anderen sendet es diese beim Speichern an die gewählten Felder in Lightning. In diesem Beispiel würde also
@Todo Dokumentation überarbeiten
@Due 10-31-2008 12-00
@Prio 1

in einem Emailbody erkannt und ein entsprechender Task angelegt.
Eine weitere Idee ist die Einführung der Kennzeichnung von erkannter Information direkt in der Email. So könnten dem Benutzer beim Lesen der Email die erkannten Informationsbausteine direkt farbig (wie mit einem Textmarker) markiert werden. Ein Klick-Bereich in der Email, um direkt von dort aus mit einem Klick auf die markierten Informationen das Speichern zu veranlassen, ist ebenfalls denkbar. Dies sollte am Besten aber im Vorfeld mithilfe eines Benutzertests bezüglich der Useability evaluiert werden.

7.3 Ressourcen

In diesem abschließenden Kapitel sind eine Reihe an Online-Ressourcen aufgelistet, die mir bei der Umsetzung meines Extensions wertvolle Tipps und Unterstützung gegeben haben. Da es sich dabei

nicht direkt um Literatur handelt habe ich mich dazu entschlossen, sie an dieser Stelle aufzulisten:

Getting started with extension development
via http://kb.mozillazine.org/Getting_started_with_extension_development
Eine gute Einführung in die Extension-Entwicklung. Hier gibt es auch die Möglichkeit, sich ein entsprechendes Paket für eine Extension generieren zu lassen um somit direkt ein Startgerüst zu haben.

Mozilla Development Center
via https://developer.mozilla.org/
Die Anlaufstelle um einen Überblick über verfügbare Technologien oder spezielle Themen wie Sicherheit und Localization zu bekommen. Befindet sich stellenweise aber leider noch im Aufbau.

XUL-Planet
via http://www.xulplanet.com/
Auch hier gibt es gute Tutorials zum Einstieg in die Oberflächenprogrammierung mit XUL. U.A. wird aber auch die Funktionsweise von Event-Listener aufgezeigt. Die Seite bietet außerdem eine Vielzahl an praktischen Beispielen.

Mozilla Cross-Reference
via http://mxr.mozilla.org/mozilla/
Diese Seite bietet einen umfassenden Einblick in den Quellcode von Thunderbird und Lightning. Man kann sich die Definitionen von verschiedenen Objekten sowie die Verfügbarkeit und Arbeitsweise von Funktionen anschauen um diese ggf. weiter zu verwenden.

Zusätzlich ist folgende Newsgroup sehr empfehlenswert, hier wird gerne und gut geholfen wenn die Entwicklung mal stockt:

news.mozilla.org
insbesondere m.s.calendar *und* m.d.a.calendar

Literaturverzeichnis

[FSA07] Thomas Franz, Steffen Staab, and Richard Arndt, *The x-cosim integration framework for a seamless semantic desktop*, K-CAP 2007 – Proceedings of the Fourth International ACM Conference on Knowledge Capture (Whistler, BC), 10 2007.

[ICC+05] Neil Ireson, Fabio Ciravegna, Mary Elaine Califf, Dayne Freitag, Nicholas Kushmerick, and Alberto Lavelli, *Evaluating machine learning for information extraction*, ICML (Luc De Raedt and Stefan Wrobel, eds.), ACM International Conference Proceeding Series, vol. 119, ACM, 2005, pp. 345–352.

[KBKS07] Max Van Kleek, Michael S. Bernstein, David R. Karger, and M. C. Schraefel, *Gui — phooey!: the case for text input*, UIST (Chia Shen, Robert J. K. Jacob, and Ravin Balakrishnan, eds.), ACM, 2007, pp. 193–202.

[KLDK06] Nicholas Kushmerick, Tessa A. Lau, Mark Dredze, and Rinat Khoussainov, *Activity-centric email: A machine learning approach*, AAAI, AAAI Press, 2006.

[MEH04] Luke Mcdowell, Oren Etzioni, and Alon Halevy, *Specifying semantic email processes*, In WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, 2004.

[SHD08] Simon Scerri, Siegfried Handschuh, and Stefan Decker, *Semantic email as a communication medium for the social semantic desktop*, ESWC (Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, eds.), Lecture Notes in Computer Science, vol. 5021, Springer, 2008, pp. 124–138.