Intelligente Email Assistenten

Ausarbeitung zum Seminar

„Intelligente Interface Agenten“

Timo Krause

Andreas Vangerow

WS 2001/2002

Inhaltsverzeichnis

Intelligente Interface Agenten
E-Mail-Assistenten

1. Einleitung

In den letzten Jahren versuchten viele Forscher im Bereich der Mensch-Maschine Kommunikation neue Innovationen zu entwickeln. Ein Ansatz dieser User-Interfaces sind die „Intelligenten Interface Agenten“. Agenten sind Programme, die mehr oder minder autonome Aktionen ausführen, wenn die Aktionen dem Ziel des Benutzers entgegenkommen. Im Bereich der Interface Agenten unterstützen sie den Benutzer im Umgang mit dem Computer.
Ein anderer Ansatz ist es, den direkten Zugriff auf das Interface zu vereinfachen. Die Benutzung soll dabei noch intuitiver werden.
Da in beiden Bereichen sehr starke Fortschritte erzielt wurden, ist es von Interesse beide Ansätze zu vereinigen.
Im folgenden werden die 12 Sätze [Horvitz] einer solchen gemischten Benutzerschnittstelle vorgestellt und die Umsetzung dieser Sätze am Beispiel des E-Mail Programms LookOut demonstriert.

2. Prinzipien eines gemischten Interfaces

Die größten Probleme bei der Entwicklung eines Intelligenten Interface Agenten bestehen darin, dass der Agent nicht genau die Ziele des Benutzers kennt und nicht weiß was der Benutzer an Diensten benötigt. Weiterhin müssen Probleme der schlechten Kosten-Nutzen Vorhersage gelöst werden denn durch schlechte Einschätzungen der Kosten für eine Aktion können, für den Benutzer, wichtigere Aufgaben später ausgeführt werden, als gewünscht.

Um diese Probleme zu lösen müssen die folgenden Sätze [Horvitz] beachtet werden:

(1) Die Ergebnisse von automatisierte Diensten, die der Agent ausführt, müssen das selbe Ergebnis liefern, als wenn der Benutzer die selben Aktionen selbst ausführt.

(2) Es ist für einen Agenten meist nicht ersichtlich welche Ziele der Benutzer hat und wo er sich gerade drauf konzentriert. Ein Agent sollte ein Modell der Benutzeraufmerksamkeit pflegen um die Aktionen benutzerspezifisch zu starten.

(3) Der Agent muss die Aufmerksamkeit des Benutzers beachten, wenn er Dienste automatisch ausführt, um die Kosten und den Nutzen für den Benutzer berechnen zu können.

(4) Ein Agent, der nicht genau die Ziele des Benutzers kennt und eine Aktion ausführt, sollte vorher abwägen, wie viel die Aktion den Benutzer näher zum Ziel führt, um eventuell andere Aktionen auszuführen.

(5) Wenn sich der Agent nicht sicher über die Ziele des Benutzers ist, sollte er ein Dialogfenster öffnen und den Benutzer die Entscheidung überlassen. Dabei muss aber beachtet werden, dass der Benutzer vielleicht nicht gestört werden möchte.

(6) Ein Agent der nicht genau weiß, was der Benutzer möchte macht manchmal Fehler und führt Aktionen nicht aus, die der Benutzer gerne hätte oder stört einen Benutzer durch ausführen einer Aktion. Deshalb sollte der Benutzer in der Lage sein einen Service, den der Agent ausführen sollte direkt zu starten oder einen gestarteten Service direkt zu beenden.

(7) Beim Design von Diensten für den Agenten sollte ein Augenmerk auf die Minimierung der Kosten bei schlechter Einschätzung der Ziele des Benutzers gelegt werden.

(8) Wenn sich der Agent über die Ziele des Benutzers nicht sicher ist, sollte er auch die angestrebte Funktion ausführen, aber die nur den Teil ausführen, wo er sich sicher ist. Nach der Devise weniger ist manchmal mehr.

(9) Man sollte Agenten unter der Annahme entwickeln, dass der Benutzer Ergebnisse von Agenten aufbereiten möchte.

(10) Ein Agent sollte angemessen mit seinem Benutzer kommunizieren, d.h. er sollte sich den gesellschaftlichen Erwartungen des Benutzers entsprechend verhalten.

(11) Der Agent sollte ein Arbeitsgedächtnis haben, in dem die Interaktionen mit dem Benutzer gemerkt werden. Ein System mit einem Gedächtnis kann sich benutzerspezifische Benennungen von Objekten merken.

(12) Der Agent sollte durch Beobachtung des Benutzers dazu lernen. Ein System, welches automatisierte Dienstleistungen anbietet sollte durch Beobachtung feststellen, wann die Dienstleistung angebracht wäre, um besser mit dem Benutzer zusammen zu arbeiten.

3. Das LookOut-Programm

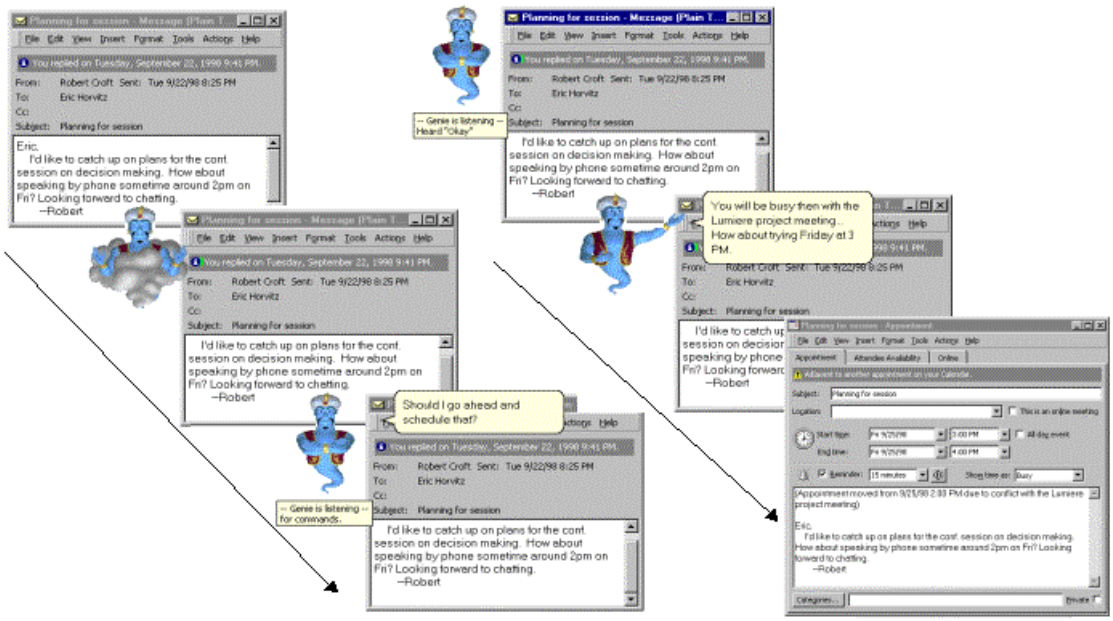
Der Hauptaugenmerk bei der Entwicklung des LookOut Agenten lag auf der Integration von automatisierten Diensten, wie Terminplanung in einem Programm, dass man nur durch direkte Manipulation ansteuern kann. LookOut ist ein Zusatz für das E-Mail- und Terminprogramm Outlook von Microsoft und wurde in den Forschungseinrichtungen von Microsoft unter der Leitung von Eric Horvitz entwickelt.

3.1. Neue Terminplanung

Wenn LookOut gestartet wird, liest es die aktuelle E-Mail ein und versucht dort ein Datum zu finden auf das sich der Absender bezieht. Dann aktiviert es den Kalender von Outlook und versucht die relevanten Felder des Termins zu füllen. Danach zeigt LookOut den Vorschlag dem Benutzer und läßt noch Änderungen an dem Vorschlag zu.
Wenn LookOut eine Nachricht bearbeitet, versucht es als erstes anhand des Erstellungsdatums und im Text gebrauchte relativer Zeitbegriffe, wie „Heute“, „Morgen“, „Gestern“ u.s.w., das gemeinte Datum heraus zu finden. Wenn aus dem Text der E-Mail kein genaues Datum ersichtlich ist, schraubt LookOut sein Ziel herunter und versucht eine Zeitspanne, auf die sich die Mail beziehen könnte heraus zu finden und stellt dem Benutzer dann einen Dialog zur Verfügung, in dem er sich entscheiden muss, wann er den Termin legt. Der Benutzer kann den Vorschlag von LookOut aber komplett über den Haufen werfen und sich für einen komplett eigenen Termin entscheiden.
LookOut kennt die typischen Muster, die in E-Mails zur Terminvereinbarung, benutzt werden. Neben der Vielfalt an Begriffen, die Leute für Zeiten und Daten benutzen, versteht LookOut auch variable Datumsangaben wie „irgendwann nächste Woche“ oder „im Laufe des Tages“, genau so wie „morgens“ oder „Nachmittags“.
Die Voranalyse von LookOut reduziert den Aufwand, den der Benutzer beim Pflegen des Adressbuches hat. Der Benutzer muss sich nicht mehr durch die Dialoge klicken und Datum, Uhrzeit und Betreff eintragen, diese Arbeit wird ihm durch LookOut abgenommen. Selbst wenn LookOut einen Fehler macht und fehlerhafte Daten einträgt ist der Zeitaufwand der Korrektur geringer.

3.2. Entscheidungsfindung in unsicheren Situationen

Der Benutzer kann LookOut direkt über den System-Tray von Windows starten. Im Hintergrund versucht LookOut die Benutzerziele zu identifizieren, indem es bereits bearbeitete E-Mails des Benutzers betrachtet. Dabei arbeitet sich LookOut durch den Header, das Subject und den Body der E-Mail und errechnet aus diesen Informationen die Wahrscheinlichkeit, dass der Benutzer seinen Terminkalender sehen will oder einen Termin eintragen will, indem ein auf Wahrscheinlichkeit basierender Klassifikator, durch Beobachtung des Benutzers im Umgang mit E-Mails, trainiert wird. Basierend auf dieser Wahrscheinlichkeit und den Kosten und Nutzen für den Benutzer, entscheidet das System, nichts zu tun und auf Benutzereingaben zu warten, eine Dialogbox zu zeigen und den Benutzer zu fragen, ob er einverstanden ist, dass LookOut die nächste Phase der Weiterverarbeitung ausführt, oder führt die nächste Phase automatisch aus.



[Bild 1. [Horvitz] zeigt LookOut im Agentenmodus. Nachdem LookOut eine ankommende Nachricht bearbeitet hat, erscheint der Agent und fragt, ob eine Aktion ausgeführt werden soll. In diesem Fall möchte LookOut die Nachricht als Termin im Kalender eintragen. Der Agent reagiert auch auf natürlich sprachliche Antworten wie „jepp“, „auf jeden Fall“ u.s.w.. In diesem Beispiel soll der Agent weitermachen. LookOut erzeugt einen neuen Termin und bespricht mit dem Benutzer weiteres, bevor der Termin abgespeichert wird. Wenn der Benutzer zu irgendeinem Zeitpunkt Desinteresse gezeigt hätte, in dem er die Nachricht weg klickt oder auf die Frage ob der Agent fortfahren soll „nein“ gesagt hätte, wäre der Agent ohne weitere Nachfrage verschwunden.]

3.3. Multimodale Interaktion

LookOut kann in zwei Modi arbeiten. Im manuellen Modus muss der Benutzer LookOut starten. LookOut macht aber auf sich aufmerksam, wenn es zu einem Zeitpunkt eine Aktion ausgeführt hätte, indem ein Symbol im System Tray auftaucht. Durch Klicken auf dieses Symbol wird LookOut gestartet.
Im zweiten Modus ruft LookOut automatisch die erforderlichen Fenster in Outlook auf und füllt sie mit den Informationen, die es aus der dazugehörigen E-Mail bekommen hat. Wenn noch Informationen fehlen, fragt LookOut automatisch bei dem Benutzer die erforderlichen Informationen nach, in dem es maßgeschneiderte Dialogboxen benutzt.
Man kann LookOut auch im Agenten-Modus laufen lassen. Dabei bedient sich LookOut aus dem „Agent social user-interface package“ von Microsoft in Form von animierten Figuren.
Im Agenten-Modus kommuniziert LookOut über eine Audio-Verbindung mit dem Benutzer, wobei die Interaktion mit Maus und Tastatur erheblich reduziert wird. In diesem Modus bedient sich LookOut eines Text-to-Speech Systems und eines Spracherkenners um mit dem Benutzer einen möglichst natürlichen Dialog zu führen. Dabei fragt LookOut alle nötigen Informationen ab und wartet auf eine Antwort, ohne dass eine Taste gedrückt werden muss.

3.4. Fehlerhaftes Starten von LookOut verarbeiten

Da LookOut sich nicht immer sicher sein kann, dass die ausgeführte Aktion richtig war, bzw. die Nichtausführung eines Befehls fehlerhaft ist, kann der Agent manuell aus dem System Tray gestartet werden. Wenn LookOut dem Benutzer einen Terminvorschlag aufgrund einer eingehenden Nachricht machen möchte, versucht LookOut vorsichtig auf sich aufmerksam zu machen. Es wartet eine bestimmte Zeit, die von der Wichtigkeit der anstehenden Aufgabe abhängig ist, und verschwindet nach Ablauf dieser Zeit. Wenn der Agent Zeichen erkennt, dass der Benutzer nachdenkt, wie „hmm“, „ähh“ u.s.w., wird die Wartezeit verlängert.

3.5. Intention und Ziel des Benutzers bestimmen

Um dem Benutzer möglichst viel und komplexe Arbeit ab zu nehmen, ist es wichtig die Unsicherheit des Systems über die Ziele des Benutzers zu betrachten. Dabei ist es schwierig einen Algorithmus zu verwenden, der einer bestimmten Wahrscheinlichkeit, dass der Benutzer eine Aufgabe erledigt haben möchte, explizit einer bestimmten Intention des Benutzers zuordnet. LookOut bestimmt Wahrscheinlichkeiten einer Benutzerintention, in dem es sich einer linearen

„Support Vector Machine“ Textklassifikation bedient [Dumais][Platt J.]. Wenn eine eintreffende Nachricht das erste Mal von dem Benutzer gelesen wird, soll sie in zwei mögliche Kategorien klassifiziert werden: „Der Benutzer will für diese Nachricht einen Eintrag in seinen Kalender erstellen.“ oder „Der Benutzer will für diese Nachricht keinen Eintrag in seinen Kalender erstellen.“. Der eingebaute Textklassifikator wurde bei Auslieferung mit 1000 Nachrichten getestet. 500 der Nachrichten sind für den Benutzer uninteressante Nachrichten und 500 Nachrichten, die für den Benutzer interessant sind und in denen eine Terminabsprache gefordert wird.

3.5.1. Von der Überzeugung zur Aktion

Wenn der Agent vor der Entscheidung steht einen Service auszuführen oder nicht, so muss er abwägen ob es für den Benutzer nützlicher ist den Service auszuführen oder nicht. Ein automatischer Service sollte erst ausgeführt werden, wenn diese „Nützlichkeit“, eine Aktion auszuführen, für den Benutzer größer ist als die „Nützlichkeit“ nichts zu tun. Der Agent muss dabei die Kosten, den Nutzen und die Unsicherheit über das Ziel des Benutzers in Betracht ziehen.

3.5.2 Aktionen, Intentionen und Ergebnisse

Die Wahrscheinlichkeit p(G|E), dass der Benutzer ein bestimmtes Ziel (G, Goal) hat, errechnet LookOut, indem es den Benutzer im Umgang mit einer neuen Nachricht beobachtet.
Um eine Entscheidung treffen zu können, ob eine Aktion ausgeführt wird, oder nicht muss man vier Ergebnisse betrachten: Der Benutzer möchte entweder bei einer eintreffenden Nachricht gefragt werden, wie mit der Nachricht verfahren wird (hat das Ziel G (Goal)) oder möchte nicht gefragt werden (hat nicht das Ziel nG (not Goal)). Für jede der beiden Möglichkeiten kann LookOut in Aktion (A) treten, oder keine Aktion ausführen (nA). Jeder der vier möglichen Ergebnisse wird als „Nützlichkeit“ u bezeichnet. Die Werte liegen zwischen null und eins und werden wie folgt definiert:

u(A,G): Die Nützlichkeit eine Aktion auszuführen, wenn das Ziel G wahr ist.
u(A,nG): Die Nützlichkeit eine Aktion auszuführen, wenn das Ziel G nicht wahr ist.
u(nA,G): Die Nützlichkeit eine Aktion nicht auszuführen, wenn das Ziel G wahr ist.
u(nA,nG): Die Nützlichkeit eine Aktion nicht auszuführen, wenn das Ziel G nicht wahr ist.

Die Ergebnisse werden in folgender Tabelle nochmal dargestellt:

	Gewünschtes Ziel	Nicht gewünscht
Aktion	**u(A,G)**	**u(A,nG)**
Keine Aktion	**u(nA,G)**	**u(nA,nG)**

Tabelle 1. Vier Nützlichkeiten u ergeben sich bei der Entscheidung, ob ein Agent gestartet werden soll oder nicht.

Die einzigen Hinweise, die LookOut bekommt, erhält es durch Beobachtung des Benutzers im Umgang mit einer Nachricht. Die erwartete Nützlichkeit, eu(A/E), bei gegebenen Hinweisen, eine Aktion automatisch auszuführen wird durch Kombination der Nützlichkeit, dass der Benutzer einen Service haben möchte, mit der Nützlichkeit, dass der Benutzer diesen Service nicht haben möchte. Beides wird mit den jeweiligen Wahrscheinlichkeiten der Ergebnisse p(G|E) und p(nG|E) multipliziert.

$$eu(A/E) = p(G|E)\, u(A,G) + p(nG|E)\, u(A,nG) \qquad (1)$$

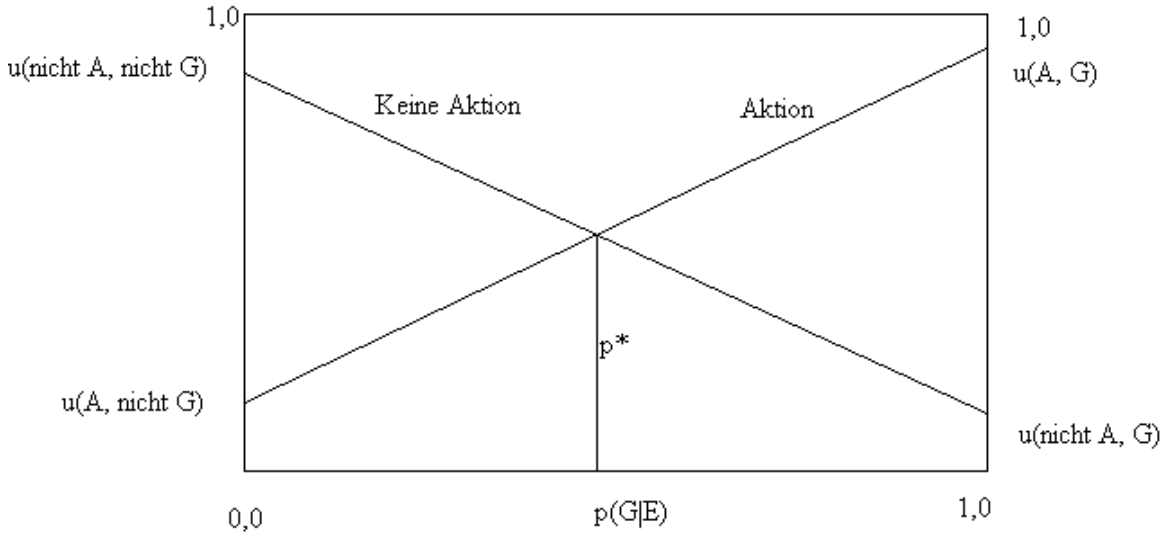
Dies kann man umschreiben, da p(G|E) = 1 – p(nG|E) ist. Also ergibt sich folgende Nützlichkeit:

$$eu(A/E) = p(G|E)\, u(A,G) + [1 - p(G|E)]\, u(A,nG) \qquad (2)$$

Äquivalent dazu ist die Nützlichkeit die automatische Aktion nicht auszuführen:

$$eu(nA/E) = p(G|E)\, u(nA,G) + [1 - p(G|E)]\, u(nA,nG) \qquad (3)$$

Visualisiert sieht das ganze folgendermaßen aus:



[Bild 2. [Horvitz]. Graphische Analyse der erwarteten Nützlichkeit einer Aktion gegenüber der nicht ausführen der Aktion, ergibt einen Schwellenwertwahrscheinlichkeit, bei deren Überschreitung die Aktion ausgeführt wird.]

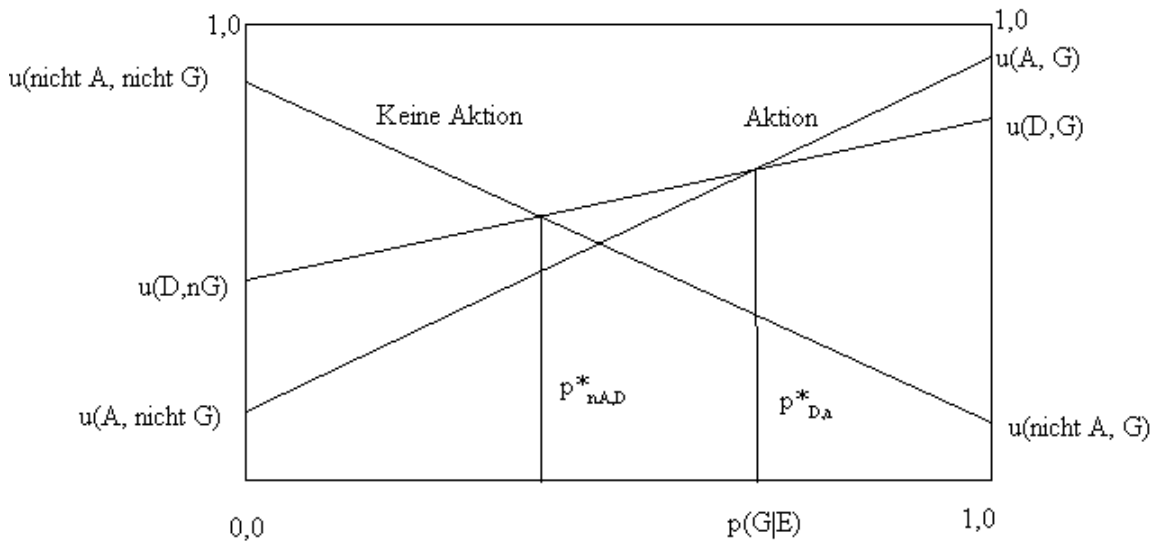
3.5.3. Erwartete Nützlichkeit und Schwellenwert für Aktionen des Agenten

Der Ort an dem die Linien in Bild 2, die die erwartete Nützlichkeit repräsentieren, kreuzen lassen auf die Wahrscheinlichkeit folgern, dass der Benutzer das gefragte Ziel hat. An dieser Stelle p*, wo die Nützlichkeit eine Aktion auszuführen genauso groß ist wie die Nützlichkeit die Aktion nicht auszuführen, ist der Schwellenwert, an dem das System entscheidet, ob es die Aktion ausführt. Wie man an dem Graphen in Bild 2 erkennt.
Das System wertet die Nützlichkeiten aus, in dem es alle vier berechneten Nützlichkeiten mit dem Schwellenwert vergleicht und entscheidet anhand des Vergleiches, ob es für den Benutzer von Interesse ist den Service zu starten.

3.5.4. Die Dialogbox als Aktion

Anstatt nur unzureichend zu raten, ob der Benutzer den Service automatisch ausgeführt haben möchte, gibt es natürlich noch die Möglichkeit den Benutzer zu fragen. Man kann ziemlich einfach die Nachfrage durch eine Dialogbox in das bestehende Konzept einbauen, in dem man die Nützlichkeit berechnet, bei dem Benutzer nach zu fragen. Man betrachtet die zwei zusätzlichen Nützlichkeiten: Der Agent fragt den Benutzer nach seinem Ziel und der Benutzer möchte darüber auch nachgefragt werden, u(D,G), und der Benutzer möchte nicht gefragt werden, u(D,nG).

Das nächste Bild zeigt eine neue Linie, die die Nützlichkeit darstellt, bei dem Benutzer nach zu



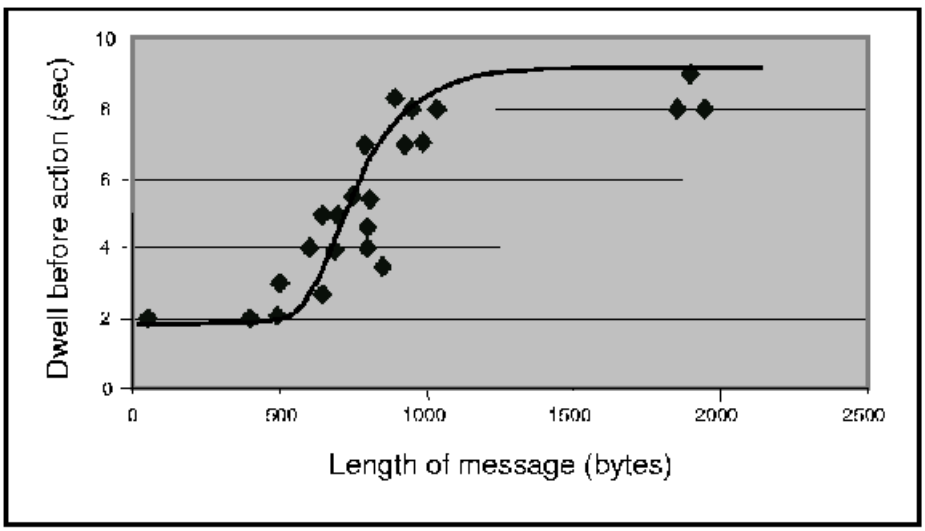
fragen.

[Bild 3.[Horvitz] Durch Betrachtung der Nützlichkeit ein Dialogbox einzusetzen ergeben sich zwei Schwellenwertwahrscheinlichkeiten, bei deren Überschreitung keine Aktion, eine Dialogbox oder eine sofortige Aktion ausgeführt wird.]

Wie man aus dem Graphen entnehmen kann ist die typische Nützlichkeit bei dem Benutzer nach zu fragen, wenn dieser nicht gefragt werden möchte (nG), ist größer als die Nützlichkeit, dass eine Aktion ausgeführt wird, obwohl der Benutzer dies nicht möchte. Genauso, wie die Nützlichkeit kleiner ist bei einer gewünschten Aktion noch mal nach zu fragen, als sie einfach so aus zu führen. Wie man erkennt ergeben sich dabei zwei neue Schwellenwerte. Der Schwellenwert zwischen keine Aktion zu starten und bei dem Benutzer nach zu fragen und der Schwellenwert zwischen dem nachfragen und die Aktion einfach zu starten.
Im LookOut System ist es dem Benutzer möglich die default Werte für die Nützlichkeiten und die Schwellenwerte manuell einzustellen.

3.6. Einsatz von Aktionen in Abhängigkeit der Benutzeraufmerksamkeit

Automatisch eine Aktion auszuführen zu einem Zeitpunkt, wo der Benutzer noch nicht bereit dazu ist, kann den Benutzer stören, genauso, wie es störend ist auf eine Aktion des Systems zu warten. Deshalb wurde in LookOut ein Mechanismus verwendet der die Benutzeraufmerksamkeit beobachtet.
Bei der Entwicklung von LookOut wurden Benutzer beobachtet, die eine neue Nachricht bekommen und die Zeit gemessen, bevor sie eine Aktion starten, wie zum Beispiel einen Termin in den Kalender ein zu tragen. Diese Daten wurden in ein Datenmodell eingearbeitet, dessen sich LookOut bedient. Dabei wurde eine nicht-lineare Beziehung zwischen der Länge einer Nachricht und dem starten des Service beobachtet, die sich am besten durch eine sigmoide Funktion darstellen lässt. Einige Testergebnisse sind in der nächsten Zeichnung dargestellt.



[Bild 4.[Horvitz. Verhältnis von Länge einer Nachricht in Bytes zu der Betrachtungszeit des Benutzers.]

LookOut kann sich dieses Modell auch selbst erarbeiten, in dem man es im Lernmodus startet und

es dann die Zeit misst, die der Benutzer zum lesen braucht bevor er eine Aktion startet. Diese Daten werden dann für jeden Benutzer aufgezeichnet, so dass jeder seine personalisiertes LookOut hat. Es ist auch möglich diese Zeit manuell in LookOut einzutragen.

3.7. Lernen durch Beobachtung

Wie in Kapitel 6 beschrieben, lernt LookOut. Der Mechanismus der dafür verantwortlich ist, beobachtet den Benutzer im Umgang mit Nachrichten und speichert die relevanten Daten ab. LookOut entwickelt ein benutzerspezifisches Muster von Nachrichten, die der Benutzer in seinen Kalender einträgt, und die der Benutzer nicht eintragen will, so dass LookOut dies automatisch machen kann. Wenn die Kalenderfunktion von Outlook gestartet wird, merkt sich LookOut die Nachricht als wichtig und merkt sich zusätzlich die Zeit, die der Benutzer benötigt hat, bevor die Kalenderfunktion gestartet wurde.

8. Fazit

Wenn man einen intelligenten Agenten entwickeln möchte, der effektiv mit einem Benutzer zusammen arbeiten soll, so sollten die Prinzipien des gemischten Interfaces [Horvitz] beachtet werden. Diese Prinzipien wurden dann am Beispiel des LookOut Programms gezeigt. Bei der Entwicklung wurde besonders auf die Entscheidungsfindung in unsicheren Situationen, die verschiedenen Interaktionsmöglickeiten mit dem System, die Vermeidung von Fehlern geachtet. Weitere Augenmerke lagen bei der Bestimmung der Benutzerintention, der Auswertung dieser Intention um Aktionen aus zu führen und der Ausführung der Aktionen in Abhängigkeit der Aufmerksamkeit des Benutzers. Als letztes wurde noch mal kurz der Lernmechanismus von LookOut skizziert.

4. Theorie intelligenter Email-Systeme in bezug auf Re:Agent

4.1.Einführung

Ohne Zweifel lebt die heutige Gesellschaft in einem Informationszeitalter. Die Abhängigkeit von den elektronischen Medien ist nicht nur in der Industrie und im Gewerbe sehr groß, sondern auch in den privaten Haushalten. Allein die große Masse an Informationen im Internet zeugt von dieser Entwicklung. Folglich wird die spezifische Suche nach einer bestimmten Information erschwert

[Boone, 1998, 1. Introduction].
Die Entwickler des Email-Systems Re:Agent haben einen Agenten entwickelt, der dem User bei seiner Suche nach einer spezifischen Information helfen soll. Die Zielgruppe für Re:Agent bilden Individuen und Institutionen, die regelmäßig hohe Informationsmengen verarbeiten müssen.
Der Lösungsansatz für das Problem der spezifischen Suche ist nach Boone ein automatisches System, das Medien wie Emails, Fax, voicemail, webpages nach benutzerdefinierten Prioritäten sortiert. Dieser Lösungsansatz entspricht dem „Intelligent Information Management".

4.2. Intelligent Information Management

Intelligent Information Management ist definiert als ein automatisches System für elektronische Medien (web pages, email, newsgroup articles, ...), das nach den Interessen der Benutzer agiert [Boone, 1998, 1.1 Intelligent Information Management]. Das System basiert auf:

I. Artifical Intelligence
II. Machine Learning
III. Information Retrieval

4.3. Email Filtering

Durch die Verwendung von „Email Filtering" setzen die Entwickler von Re:Agent die Idee des „Intelligent Information Management" um. Ihr Agent nutzt die Struktur innerhalb einer Mail, um eine passende Einordnung innerhalb der vorhandenen Ordner zu treffen, in denen der User seine Emails ablegt. Zu dieser Struktur zählen der Absender, der Betreff und der Inhalt, der durch die Erkennung von Schlüsselworten bestimmt wird. Für die Erkennung von Schlüsselwörtern wird herkömmlich die „Nearest Neighbor Approach" Methode angewandt, die die Entwickler von Re:Agent optimiert haben. Die verbesserte Methode wurde von den Entwicklern „Concept Features Approach" genannt. Wir werden uns zunächst der „Nearest Neighbor Approach – Methode" zuwenden.

4.3.1. Nearest Neighbor Approach

Der Email-Agent, der diese Methode der Schlüsselworterkennung benutzt, muß die aktuelle Email mit anderen Emails vergleichen, die er bereits eingeordnet hat, und schlägt die wahrscheinlichste Einordnung vor. Die Realisierung läßt sich in sechs Schritte aufteilen.
1. Schritt: Der Text der Email wird nach Schlüsselworten durchsucht, die dem Agenten

bereits bekannt sind.

2. Schritt: Die Anzahl der Übereinstimmungen wird für jedes Wort hochgezählt.

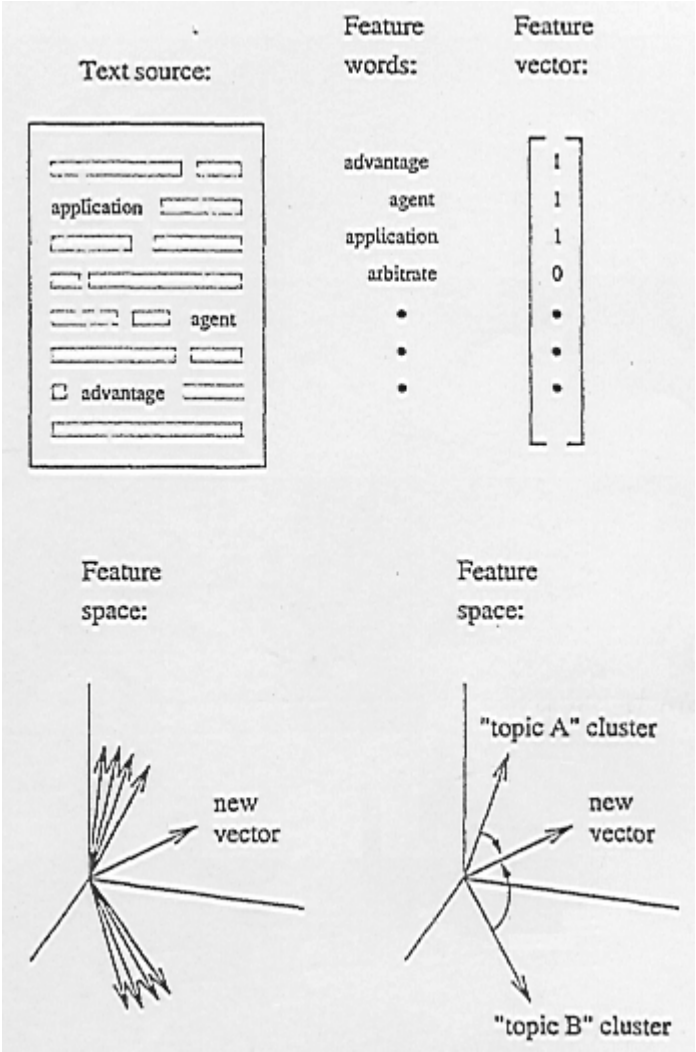
3. Schritt: Es wird ein Vektorraum gebildet, dessen Dimension der Anzahl der gefundenen Schlüsselworten gleicht. Das bedeutet, dass im Falle von n Schlüsselworten ein R^n existiert [Boone, 1998, 1.3 Email Filtering].

4. Schritt: Die gefundenen Ergebnisse, die als Vektoren auch „feature vectors“ genannt aufgefaßt werden, werden in einem resultierenden Vektor zusammengefasst.

5. Schritt: Die Vektoren von Emails mit der gleichen Einordnung sind auch in einem Vektor zusammengefasst. Dann wird ein Vergleich zwischen diesen Vektoren und dem Vektor getroffen, der die neue einzusortierende Email repräsentiert.

6. Schritt: Der Agent weist anhand des Winkels zwischen dem Vektor neuen einzusortierenden Email und den übrigen Einordnungsvektoren, die Email einer Ablage zu. Je kleiner der Winkel zu einem Einordnungsvektor ist, desto wahrscheinlicher ist die Übereinstimmung. Daher wird dieser Ablage, die durch den entsprechenden Einordnungsvektor repräsentiert wird, die neue Mail zugeordnet.

Der Nachteil dieser Methode ist, dass die Anzahl der Schlüsselwörter und die Schlüsselwörter selbst sehr sorgfältig und begrenzt gewählt werden müssen. Die Begründung liegt darin, dass die Effizienz eines lernfähigen Algorithmus weitaus geringere Dimensionen benötigt.



[Bild 5: Nearest Neighbor Approach, Quelle: Boone, 1998]

4.3.2. Concept Features Approach

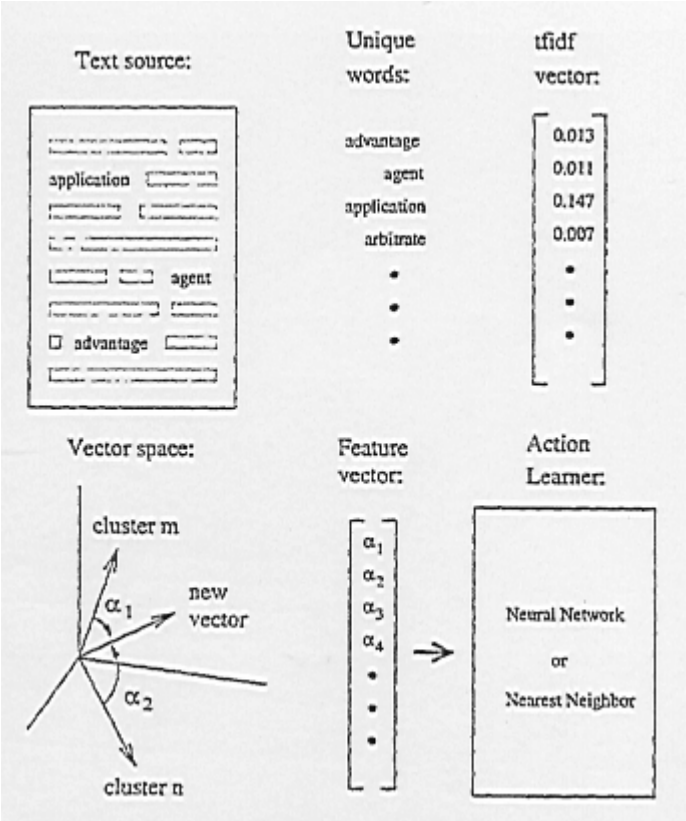
Im Gegensatz zur „Nearest Neighbor Approach – Methode“ lernt Re:Agent die spezifischen Strukturen von den bereits sortierten Emails automatisch und während der Benutzer seine Emails sortiert. Folglich nutzt Re:Agent die Emails selbst um Schlüsselwörter zu generieren, was ein großer Vorteil ist, da die Anpassungsfähigkeit an individuelle Benutzer steigt. Zudem ist der „feature vector“ hier keine Aufzählung von gemeinsamen Schlüsselworten, sondern ein statistisches Maß für Gleichheit zwischen der neuen Email und den übrigen Einordnungen. Da die Merkmale, die für die Gleichheit verantwortlich sind, aus dem Aufbau der Email resultieren, werden die Merkmale auch „concept features“ genannt.

Nun stellt sich die Frage, wie die Merkmale einer Email nun überhaupt erkannt werden. Die Email wird zunächst in einen tfidf Vektor („term-frequency-inverse-document-frequency“), in Abhängigkeit von der Anzahl ihrer Wörter, umgewandelt [Boone, 1998, 2.2 Feature Extraction]:

$$tfidf_{Wortvektor} = \frac{\text{Anzahl des Wortes}}{\text{Anzahl der Dokumente in denen dieses Wort erscheint}} \quad \forall \text{ Wörter} \in \text{Email}$$

Die tfidf Vektoren dienen nur zur Repräsentation einer Email. Die Realisierung ist hier in fünf wesentliche Schritte aufgeteilt.

1. Schritt: Ähnliche Emails werden durch ihre vom User bestimmte Einordnung in Gruppen zusammengefasst und ihre gemeinsamen Informationen werden als Merkmale verwendet.
2. Schritt: Ein statistisches Maß für Gleichheit zwischen der neuen Email und den übrigen Einordnungen wird getroffen.
3. Schritt: Es wird ein Vektorraum gebildet, in dem die Einordnungsvektoren und der neue tfidf Vektoren eingetragen werden.
4. Schritt: Der Agent misst die Winkel zwischen dem neuen tfidf Vektor und den bestehenden Einordnungsvektoren. Auch hier richtet sich die Zuordnung nach der Größe des Winkels. Aber zusätzlich werden die Winkel an den Algorithmus weitergegeben, der die Lernfähigkeit steuert. Dadurch wird eine Optimierung vollzogen. Die Lernfähigkeit operiert einzig und allein auf diesen Winkeln.
5. Schritt: Um eine Beziehung zwischen den „feature vectors“ und den „action vectors“ festzulegen, die letztendlich zum Beispiel der Einsortierung dient, wird eine numerische Näherungsfunktion verwendet.



[Bild 6: Concept Features Approach, Quelle: Boone, 1998]

4.4.Anforderungen an den User

Bei Re:Agent hat der User zwei Möglichkeiten Merkmale zu definieren. Zum einen automatisch und zum anderem manuell.

Bei der automatischen Definition von Merkmalen gibt der User dem Agenten die Anweisung, alle neuen Emails in einen bestimmten Ordner einzufügen. Die Voraussetzung ist jedoch, dass in diesem Ordner bereits vom User sortierte Emails vorhanden sind. Der Agent erstellt bei diesem Vorgang eine Struktur aus Merkmalen basierend auf den Emails in diesem Ordner und vergleicht sie mit den neuen Emails. Alle Emails, bei denen Übereinstimmungen mit diesen Merkmalen auftreten, werden in diese Ablage einsortiert.

Bei der manuellen Definition stattet der User den Agenten mit Schlüsselwörtern aus und muß Beispiel-Emails verfassen.

Der User muß zudem Emails in bestimmte festgelegte Kategorien einteilen, damit der Lernalgorithmus optimal arbeiten kann. Folgende Kategorien müssen vorhanden sein: *high priority, low priority, social, annoucements* [Boone, 1998, 2.1 Agent Architecture].

4.5.Bewertung

Source Folder	*Description*	*Number of Keywords*	*Number of Examples*	*Task Category*
papers	conference paper announcements	23	28	work
acrobot	acrobot robot discussion	35	50	work
agents	intelligent agents discussion	18	15	work
ibm	communications with IBM	29	54	work
cga	messages from Dr. Atkeson	18	34	work
jkh	messages from Dr. Hodgins	12	7	work
abowd	messages from Dr. Abowd	10	4	work
learning	discussions about machine learning	55	24	work
internships	Summer internship information	30	40	work
social	email about social events	28	254	other
spam	unsolicited commercial email	42	150	other
tara	email from family member	26	75	other
daily	other daily notes	35	475	other

[Bild 7: Testergebnisse von Re:Agent, Quelle: Boone, 1998]

Um Re:Agent zu testen haben die Entwickler zehn verschiedene vorhandene Emailablagen (Source Folder) benutzt. Die Aufgabe von Re:Agent war nun diese zehn Emailablagen in die beiden Kategorien „work“ und „other“ einzuordnen. Zweidrittel der vorhanden Emails wurden zum Trainieren eingesetzt und der Rest zum Testen (Number of Examples). Obwohl die Anzahl der Schlüsselworte gemessen an der Gesamtanzahl der Worte eher gering ist, hat Re:Agent die Emails den richtigen Kategorien zuordnen können.

5. Mailcat

5.1. Einführung

Mailcat ist ein intelligenter Agent, der an dem IBM Thomas J. Watson Research Center als add-on für den Lotus Notes mail client entwickelt wurde. Das Besondere an Mailcat ist nun, dass er den zeitlichen und kognitiven Aufwand, der benötigt wird um Emails einzusortieren, stark reduziert, was das primäre Ziel der Entwickler war [Segal, Kephart, 1999, 1. Introduction].

5.2. Eigenschaften von Mailcat

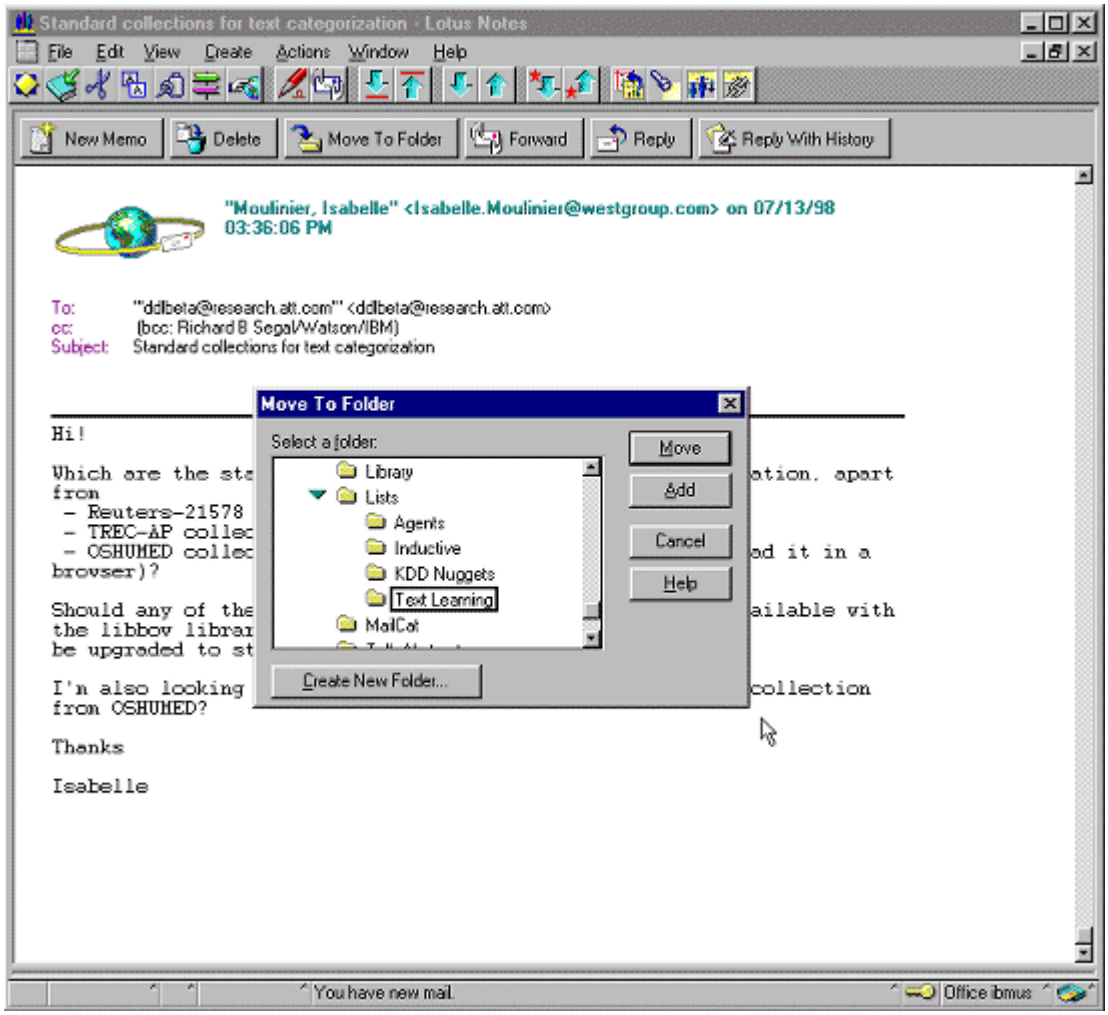
Bei der Installation von Mailcat werden alle vorhandenen Emailablagen analysiert und ein Textklassifizierer bestimmt die Usercharakteristik in bezug auf seine Einordnung [Segal, Kephart, 1999, 1. Introduction]. Mailcat wird dann dem User drei Einordnungen vorschlagen, die der User dann durch einen shortcut Button bestätigen kann. Unabhängig von der Userauswahl, wird Mailcat immer aktualisiert, wenn eine Email zugeordnet wird.

5.3. Design und Idee von Mailcat

Bei der Entwicklung des Email-Agenten Mailcat sollte in erster Linie ein fundamentales Designprinzip vorherrschen und zwar Einfachheit. Damit wollten die Entwickler folgendes erreichen: Der User sollte keine Einarbeitungszeit benötigen und Mailcat benutzen können ohne zu merken, dass Mailcat läuft.

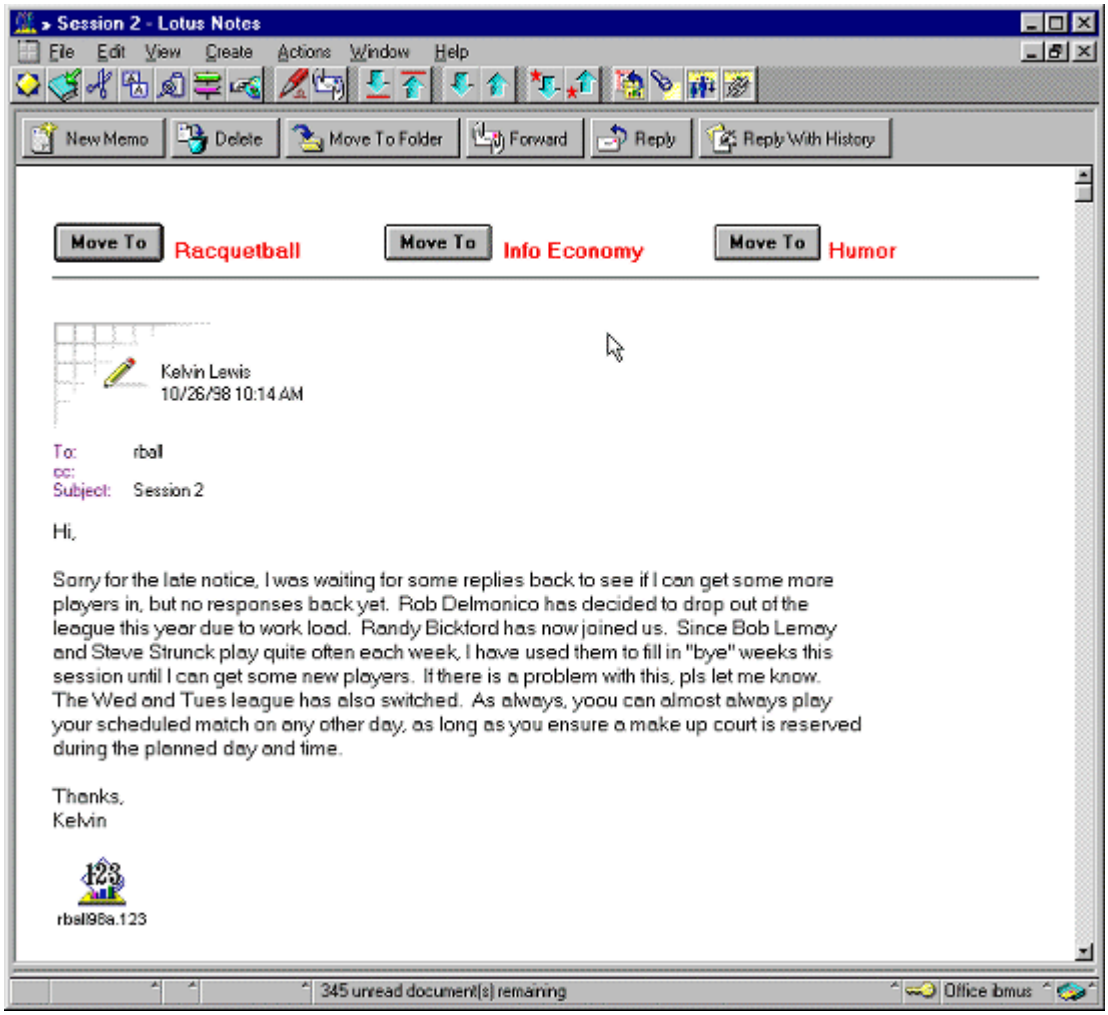
5.3.1. Beispiel Lotus Notes

Der Lotus Notes mail client basierte auf einer Ordnerhierarchie, in der die Emails abgelegt wurden. Das Problem war nun, dass um eine Email ablegen zu können, jedes Mal ein Fenster aufgemacht werden musste. Zudem musste er durch einen Mausklick jedes Mal ein Ordner öffnen, um eine Email darin abzuspeichern. Wenn der User nun mehrere Ablagen besaß, musste er die „Scroll-Leiste“ betätigen. Die Benutzerfreundlichkeit wurde somit stark eingeschränkt [Segal, Kephart, 1999, 2. Overview of Mailcat].



[Bild 8, Quelle: IBM Research Center]

Durch dem Einsatz von Mailcat wird die Benutzeroberfläche vereinfacht. Mailcat erzeugt drei Vorschläge in Form von „MoveToFolder“ Buttons über jeder geöffneten Email. Der linke Button ist die vom Agenten berechnete höchste Wahrscheinlichkeit, der mittlere ist die zweithöchste, der rechteste die dritthöchste Wahrscheinlichkeit. Die Buttons erlauben dem User eine schnelle Zuordnung, ohne eine Bestätigung zu fordern. Sollte man eine Email falsch einsortieren, ist eine Korrektur durch den „Reply“ Button möglich. Man sieht sehr deutlich, dass das Entwicklungsziel, den zeitlichen und kognitiven Aufwand der Einsortierung stark zu reduzieren, erreicht wurde.



[Bild 9, Quelle: IBM Research Center]

5.4. Textklassifizierung bei Mailcat

Petty Maes sagte aus, dass das nötige Training eine Barriere an Aufwand und Zeit setzt, die die intelligente Emailverarbeitung hemmt [Segal, Kephart, 1999, 2. Overview of Mailcat].. Maes Lösung hieß „Kollaboratives Lernen“, d.h. dass mehrere Agenten existieren, die gemeinsam an der Lösung eines Problems arbeiten. Vorraussetzung ist dabei die Fähigkeit der Agenten untereinander kommunizieren zu können.

Die Entwickler von Mailcat haben eine alternative Lösung gefunden. Sie benutzen Textklassifizierung um jede Email passend zuzuordnen. Die Textklassifizierung muß jedoch mit bereits abgelegten Emails trainiert werden, um die Usercharakteristik zu lernen. Mailcat verwendet ein TF-IDF Textklassifizierer, der bei einer Menge von 1000 Emails auf einem Pentium II 400 Mhz ca. 4min benötigt.

5.4.1. Probleme der Klassifizierung

Ein Problem der Klassizifizierung von Emails ist es, dass sich die Ablagen eines Users im Laufe ihrer Benutzung verändern. Die Ordner werden teilweise reorganisiert und gelöscht. Die Entwickler von Mailcat benutzen daher einen Klassifizierer, der „incremental learning“ (Erweiterndes Lernen) unterstützt [Segal, Kephart, 1999, 2. Overview of Mailcat]. Das besondere an ihrem Klassifizierungsmodel ist, dass es durch neu hinzugefügte und gelöschte Emails auch stets aktualisiert wird. Eine weitere Eigenschaft ist, das die Kosten an Zeit und Rechenaufwand linear im Verhältnis zur Länge bleiben. Aufgrund der eher geringen Kosten braucht Mailcat kein periodisches „re-training“ oder ein „overnight re-training“, wie andere Email-Agenten, die einen „batch learning algorithmus“ benutzen. Das bietet dem User einen Vorteil in der Anwendung von Mailcat: Der User erstellt einen neuen Ordner und ordnet dort im Laufe der Benutzung wenige Emails ein. Mailcat kann dann innerhalb diesem Benutzungszeitraum diesem Ordner sofort Emails zuordnen.

5.4.2. Mailcats Klassifizierungs-Algorithmus

Mailcats Entwickler modifizierten einen Textklassifizierer von der Firma AIM [Segal, Kephart, 1999, 3. Text Classifier]. Der AIM Textklassifizierer repräsentiert eine Test-Email oder ein Training M als Worthäufigkeitsvektor F (M), in dem jede Komponente F (M,w) die totale Anzahl des Wortes w in M repräsentiert [Segal, Kephart, 1999, 3. Text Classifier]. Der Mittelwert-Ablagenvektor F (F, w) wird geschätzt durch die Summe aller Worthäufigkeitsvektoren für jede Email in der Ablage F [Segal, Kephart, 1999, 3. Text Classifier].

$$\mathrm{F}(F, w) = \sum_{M \in F} \mathrm{F}(M, w)$$

Die Mittelwertvektoren für jede Ablage werden in einem Index gespeichert, in dem jeder Vektoreintrag invertiert wird. Wird eine Email gelöscht, wird der dazugehörige von dem entsprechenden Mittelwertvektor abgezogen.

Ein Worthäufigkeitsvektor W (F, w) dient nun dazu, die neue angekommene Email zu klassifizieren. Die Vektoren W werden ebenfalls in einem invertierten Index gespeichert. Durch eine Metrik (Distanzfunktion) die von einem Mathematiker Salton stammt, wird die Gleichheit zwischen einer neuen Email und den Ablagen einzeln berechnet [Segal, Kephart, 1999, 3. Text Classifier].

$$SIM4(\mathcal{M},\mathcal{F}) = \frac{\sum_{w\in\mathcal{M}} F(\mathcal{M},w)\,W(\mathcal{F},w)}{\min(\sum_{w\in\mathcal{M}} F(\mathcal{M},w), \sum_{w\in\mathcal{M}} W(\mathcal{F},w))}.$$

[Bild 10: Metrik, Quelle: Segal, Kephart, 1999]

Mailcats Entwickler ließen nun die Schätzung in der Metrik weg, indem, wenn eine Email *M* klassifizert werden soll, der Klassifizierer „on the fly“ (in Echtzeit) die Menge der Worthäufigkeiten überschlägt. Daher benötigt Mailcat keinen zweiten ineffizienten Index. Durch diese Beziehung, die sich in erster Linie auf die Distanzfunktion auswirkt, wird erreicht, dass die Zeit, die benötigt wird linear bleibt, um eine neue Klassifizierung auszuführen [Segal, Kephart, 1999, 3. Text Classifier].

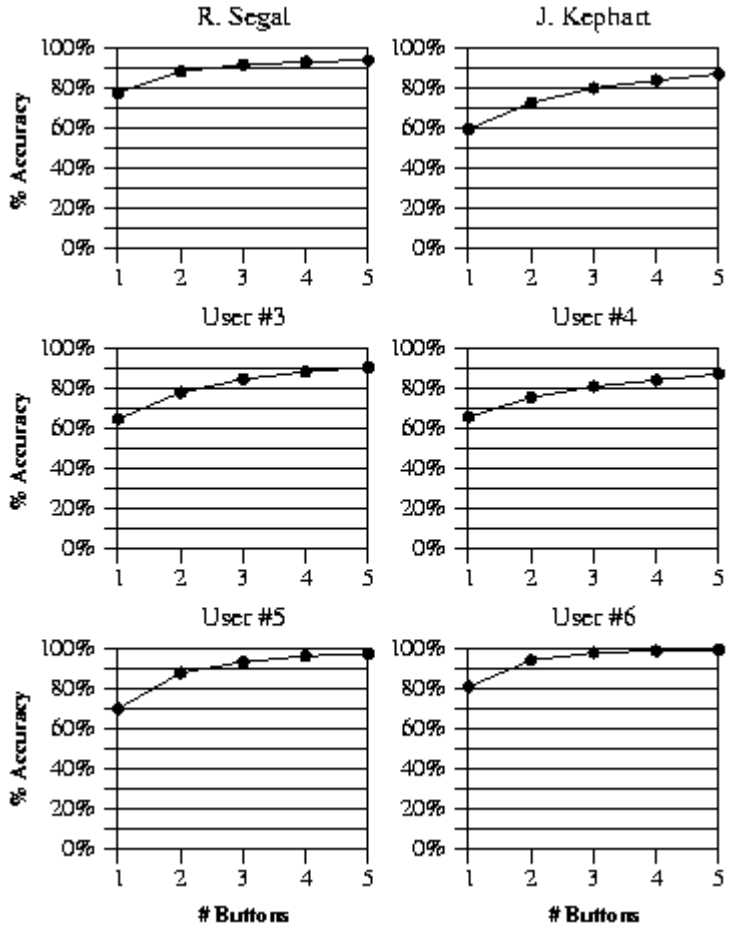
5.5. Testergebnisse

Am ersten Test von Mailcat waren sechs Personen beteiligt [Segal, Kephart, 1999, 5. Experiments].

Database	# Folders	# Messages
R. Segal	66	814
J. Kephart	56	1420
User #3	43	2433
User #4	34	473
User #5	15	553
User #6	14	3020

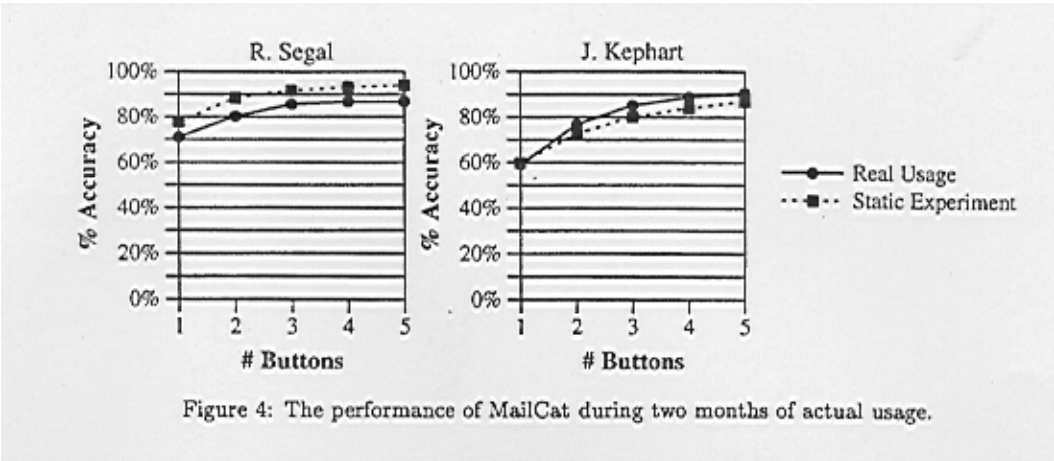
[Bild 11: Liste über die Emailablagen der sechs Versuchspersonen, Quelle: Segal, Kephart, 1999]

Wie erwartet ist die Textklassifizierung um so schwieriger, je höher die Anzahl der Ablagen ist. Der folgende Ergebnisgraph zeigt die Genauigkeit unter der Annahme von einem bis zu fünf Buttons. Die Genauigkeit von Mailcat mit n Buttons ist definiert als die Häufigkeit mit der ein „MoveToFolder“ Button den richtigen Ordner zuweist. Wie man sieht schneidet Mailcat hier sehr gut ab. Wichtig zu erwähnen ist auch, dass alle Versuchspersonen sich in der Wahl ihrer Ablage bei einem kleinen Prozentsatz der Emails individuell verhalten.



[Bild 12: Ergebnisgraph über die Genauigkeit von Mailcat bei den sechs Versuchspersonen, Quelle: IBM Research Center]

Der zweite Test ging mit zwei Versuchspersonen über zwei Monate. Es zeigt sich, dass die Ergebnisse im statistischen Bereich des ersten Tests liegen.



[Bild 13: Ergebnisgraph über die Genauigkeit von Mailcat bei dem zweiten Test über 6 Monate, Quelle: Segal, Kephart, 1999]

Abschließend läßt sich sagen, dass die Fehlerrate mit drei Buttons bis zur Hälfte im Gegensatz zu einem Button reduziert wird. Mit fünf Buttons wird die Fehlerrate zwar weiterhin reduziert, aber die Benutzerfreundlichkeit sinkt, daher entschieden sich die Entwickler, Mailcat mit drei Buttons auszustatten.

5.6. Konzept von Mailcat

Das Konzept von Mailcat kann laut den Entwicklern auch in anderen Bereichen der elektronischen Organisation, wo Daten in eine Hierarchie geordnet werden können, verwendet werden. Als Beispiele führen sie unter anderem bookmarks, files, audio recordings auf.

6. Schlusswort

Eine Bewertung der einzelnen Email-Agenten lassen wir hier außen vor, da wir selbst die Programme nicht testen konnten.
Neben LookOut haben wir den Agenten Re:Agent betrachtet. Re:Agent nutzt die Emails selbst um Schlüsselwörter zu generieren. Mit der verwendeten „Concept Features Approach“ Methode werden tfidf Vektoren benutzt um die Emails des Users zuzuordnen.
Mailcats Entwickler modifizierten einen Textklassifizierer. Es wird für jede Email ein Mittelwert-Ablagenvektor F (F, w) anhand der Summe aller Worthäufigkeitsvektoren geschätzt. Anschließend wird durch eine Metrik (Distanzfunktion) die Gleichheit zwischen einer neuen Email und den Ablagen einzeln berechnet. Bei Mailcat überzeugen vor allem die hohe Benutzerfreundlichkeit und die Testergebnisse.
Wie man sieht, hat sich auf dem Gebiet der intelligenten Emailverarbeitung sehr viel getan und es wird sicherlich noch viel Entwicklungsaufwand in dieses Gebiet gesteckt. Allein wenn man die Relation zwischen gesendeten Emails und verschickten Briefen betrachtet, lasst sich erkennen welche Bedeutung dem intelligenten Email Assistent zukommt.

Literaturverzeichnis

[Boone, 1998] Boone, Gary
Concept Features in Re:Agent, an Intelligent Email Agent (Seite 295 –298),
Proceedings of the 2nd Annual Conference on Autonomous Agents,
(Seite 295 –298), ACM, 1998

[Dumais, S. T., Platt, J., Heckermann, D., Sahama, M.] Dumais, S. T., Platt, J., Heckermann, D., Sahama, M.
Inductive learning algorithms and representations for text categorization.
Proceedings of CIMK98. (Bethesda MD, November 1998). ACM Press, 148-155.

[IBM Research Center] Mailcat http://www.research.ibm.com/swiftfile

[Platt, J.] Platt, J. *Fast training of SVMs using sequential minimal optimization.*
To appear in: B. Scholkopf, C. Burges, A. Smola (Eds.) Advances in Kernel Methodes – Support Vector Learning, MIT Press, 1999.

[Segal, Kephart, 1999] Segal, Richard B.; Kephart, Jeffrey O.
Mailcat: An Intelligent Assistant for Organizing E-Mail (Seite 276 –282)
IBM Thomas J. Watson Research Center, 1999
Proceedings of the Third Annual Conference on Autonomous Agents (pp. 276-282). ACM Press, 1999.