Besondere Lernleistung Informatik

Erweiterung des IMAP-Protokolls zur vereinfachten Moderation von Newsgroups und Mailinglisten

Heiko Studt

Lehrer: Herr Broelemann (Informatik-LK)

Inhaltsangabe

Themenwahl

Voraussetzung -- Hamster

Ende 1999, zu dieser Zeit hatte ich schon Internet über ein 14.4Kbit-Modem und auch einen eigenen Rechner, kam ich mit dem Medium Usenet in Berührung. Gerade einmal einen Monat nach den ersten Erfahrungen und einem Totalverlust meiner Mail-/Newsdaten, installierte ich mir einen Newsserver namens Hamster, mit dem ich bereits vorher liebäugelte.
Aufgrund von guten Erfahrungen mit Hamster und einer (technisch) sehr informativen Mailingliste, außerdem aufgrund meines gestiegenen Mail- und Newsaufkommens, verfolgte ich die Entwicklung weiter. Zu dieser Zeit war Hamster noch kein Open-Source.

Voraussetzung -- Programmierung

Etwa Mitte 2000 wurde dann der Sourcecode, welcher in „Delphi 3 Professional“ geschrieben war, vom Hamsterprogrammierer veröffentlicht. Da ich zu dieser Zeit nur eine „Delphi 3 *Standard*version“ besaß, konnte ich nicht aktiv an der Weiterentwicklung teilnehmen, außerdem hatte ich noch keinerlei Erfahrung mit der Programmierung von Win32-Programmen, GUI und Multithreading. Außerdem waren meine Kenntnisse bezüglich Pascal/Delphi recht bescheiden.
Etwa zum Herbst/Winter 2000 bekam ich Delphi 5 geschenkt, mit dem ich Hamster kompilieren konnte. Ich sammelte von nun an Erfahrungen mit verschiedenen kleineren Änderungen, hauptsächlich an der GUI; außerdem merzte ich einige Bugs aus.

IMAP

Zu Beginn des Sommers 2001 faßte ich den Plan einen IMAP-Server in den Hamster einzubauen, doch aufgrund von verschiedenen anderen Projekten konnte ich dies zeitlich nicht sofort bewerkstelligen. Zudem waren meine Erfahrungen in Delphi bei weitem noch nicht ausreichend. Daher programmierte ich einige andere Projekte, so daß sich mein Wissen erweiterte.
Anfang Herbst 2001, also in der 12/1 fing ich dann an, basierend auf den Hamster 1.3.22.10x (später 1.3.23.x), die Grundlagen für einen neuen Server in den damaligen Hamster-Source einzubauen. Dazu las ich RFC 2060, welcher das IMAP-Protokoll beschreibt. (Dieser RFC ist erst ‚jetzt‘ durch RFC 3501 abgelöst worden)
Aufgrund der vielen in dieses Projekt gesteckten Arbeit, die ich zwar später mit Martin Germann teilte, und auch weil ich einen großen Teil meiner Freizeit in dieses Projekt investiert habe, beschloß ich das Projekt als Grundlage für eine „Besondere Lernleistung“ anzumelden.

Erweiterung für Moderation

Das genauere Thema ergab sich dann eher durch Zufall:

Jürgen Helbing, Autor eines anderen großen Windows-Newsservers (MyNews), hatte am 18.01.2003 in der Newsgroup news.software.nntp den Thread „Moderation 2004: Just thinking loudly“[1] angefangen, in dem er seine Ideen über eine zukünftige Gestaltung von Moderationen, in seiner Ursprungsidee direkt über das NNTP-Protokoll, beschreibt.

Dieses Verfahren sollte den bisherigen Standard für moderierte Newsgroups (über Email) ablösen.

Als eine der kritischen Antworten auf seine Ideen machte ich einen Gegenvorschlag, der die von ihm vorgeschlagenen und eigentlich bezweckten Verbesserungen an den Oberflächen der Clients beinhaltet, aber auf das bereits bestehende Mail-System aufsetzt[2]. Außerdem ist dieser Vorschlag in weiten Teilen kompatibel zu moderierten Mailinglisten. Zur Vereinfachung stelle ich in dieser „Besonderen Lernleistung“ nur die Erweiterung für Newsgroups dar, denn für Mailinglisten würde nahezu dieselbe Struktur gebraucht.

„Besondere Lernleistung“

Diesen Ursprungsvorschlag habe ich später verbessert und stelle ihn hier, als Thema meiner „Besonderen Lernleistung“, vor. Wie bereits erwähnt habe ich seit der 12/1 starke Vorarbeit durch das Implementieren eines IMAP-Servers für Hamster geleistet, wodurch für mich eine Referenz-Implementation dieser Erweiterung erst möglich wird. Inwieweit ich diese bis zum Abgabetermin programmieren kann, kann ich nicht absehen.

Worterklärungen

RFC: „Request for comment“ =Die Standards im Internet	Spool: Zwischenspeicherort, hauptsächlich für Email/News und Drucker verwendet
Usenet: Ein weltweites Netzwerk zum Austausch von Nachrichten	Mailinglisten: Ähnlich wie Usenet, jedoch basierend auf Emails. definierter Empfängerkreis
Routing: =Über welche Stationen läuft der Verkehr der Daten/Emails	Forwarder: Leiten empfangene Emails an andere Server/Forwarder weiter
DNS: „Domain Name Service/Server“ =Löst Namen in IP-Adressen auf	PGP: „Pretty Good Privacy“ =Verschlüsselungstool, das auch signieren kann
Header: Teil jeder Email/Nachricht Hier werden Daten zB zum anzeigen gespeichert	Message-ID: Ein Teil des Headers, hat weltweit eindeutig zu sein. Bei News sehr wichtig.
Approven: „Akzeptieren“ (Fachwort) durch Eintrag eines bestimmten Headers	MIME: „Multipurpose Internet Mail Extensions“ Transportieren von Dateien mittels einer Email
Encoden: „Verschlüsseln“, aber auch Konvertieren zwischen verschiedenen Systemen	„Flag“: Statusinformation, z.B. „Gelesen“. Wird bei IMAP auf dem Server gespeichert.
„Closed Source“: Gegensatz zum „OpenSource“ =Der Source kann nicht eingesehen werden.	

[1] Message-ID: 5945e.i13200@archiver.winews.net
[2] Message-ID: b0jelm.qs.1@Heiko.Studt.dialin.t-online.de

Einführungen

IMAP

IMAP ist ein komplexes Protokoll zum Übertragen, Verwalten und Lesen von Emails. Es reiht sich damit an der Seite von SMTP und POP3 ein. Während SMTP in Reinform aber für jeden Empfänger jeweils eine feste IP und dort einen lokalen Spool benötigt, in dem eingehende Emails zwischengespeichert werden bis sie das Emailprogramm daraus anzeigt, und POP3 nur eine sehr simple Lösung zum Laden aus externen „Spools“ darstellt, wird der Emailtransfer bzw. das Lesen der Emails durch IMAP stark verbessert und verändert.

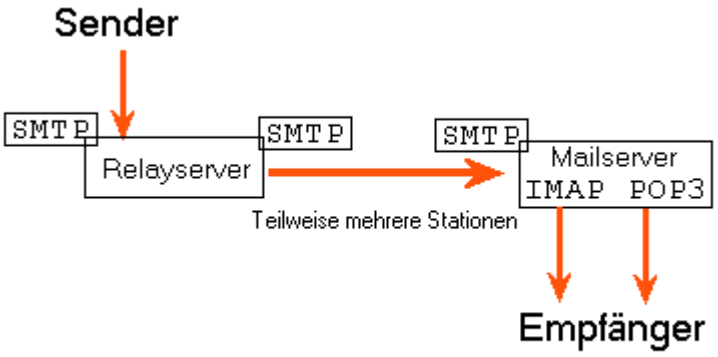
In IMAP wird nicht wie bisher die Email, die auf dem Server liegt, einfach nur zum Clienten übertragen und dieser muß schauen, was er damit anfangen und wie er die Emails verwalten kann, sondern die Verwaltung, insbesondere die der Flags (z.B. Gelesen und Beantwortet), findet direkt auf dem Server statt. Dazu wird eine Ordnerstruktur der Mailbox simuliert, je nach Serverimplementation auch real umgesetzt und die Emails beim Eintreffen dort hinein sortiert.

Beim Abruf der Emaildaten wird, je nach Client-Anforderung, die Mail ‚geparset‘, d.h. von der Serversoftware interpretiert. Diese Anforderung kann Verschiedenes beinhalten, z.B. daß der Client nur bestimmte Header, bei Attachments nur bestimmte Teile der Email, oder auch eine bestimmte Anzahl an Bytes, bekommen möchte. Auch kann er den Server anweisen in allen bzw. in Teilen die Emails zu durchsuchen.

Dadurch sollen mehrere Dinge erreicht werden:

1) Ein zentraler Server mit der Mailbox soll auch bei verschiedenen Clients dieselbe Mailbox „gleich“ anzeigen, d.h. die Clients sollen die gleichen Statusinformationen für dieselben Emails anzeigen.
2) Der Client soll möglichst entlastet und von vielen Aufgaben befreit werden. Dadurch kann der Programmierer des Clients sich auf andere Dinge, die weit wesentlicher für seine Kunden sind, konzentrieren, z.B. die „schöne“ Darstellung der Emails.
3) Auch noch so kleine „Computer“, wie zum Beispiel Handhelds, die nur geringe Speichermöglichkeiten haben, können im vollen Umfang alle Emails darstellen. Somit entfällt für den Client die Verwaltung eines gewaltigen Caches. Dies kann sogar soweit gehen, daß er nur noch Online arbeitet, also **gar keine** Informationen lokal speichern muß.
4) Mehrere Mitarbeiter/Teilnehmer/etc. können auf die gleiche Mailbox mit dem gleichen Inhalt zugreifen und ihren Inhalt verändern. Man kann somit auch öffentliche „Foren“ aufbauen, die z.B. quasi-öffentliche Kundeninformationen enthalten. Ein gutes Stichwort hierzu ist der Begriff des „Öffentlichen Ordners“.

„Routing“ von Emails



Der normale Weg vom Sender der Email hin bis zum Empfänger.

Vom Sender wird die Mail per SMTP an einen Relay-Server gesendet, dieser ermittelt per DNS den MX(Mail Exchange)-Server, für den diese Email eigentlich bestimmt ist. Dieser Server leitet dann die Email an den seiner Meinung nach zuständigen MX-Server weiter. Zuletzt wird die Email an den zuständigen Emailserver bzw., wenn der SMTP-Server sich selbst als zuständig erkennt, in das zuständige Postfach gelegt. Häufig gehen die Emails dabei über mehrere Server. Vom endgültigen Emailserver kann der Empfänger seine Emails nun per POP3 und/oder IMAP abholen. In einigen Fällen werden die Emails statt dessen direkt in ein Emailprogramm bzw. dessen „Spool“ gesendet. (Quasi ein SMTP-Only-System)

Mailinglisten (kurz)

Mailinglisten sind in der Praxis nichts anderes als Emailadressen, die alle eingehenden Emails an (mehrere) definierte Empfänger weiterleiten. In den meisten Fällen, vor allem bei „Mailinglist-Services“ wie z.B. Yahoogroups, können sie auch per Web (HTTP) angesprochen werden. Die Einrichtung und Verwaltung einer Mailingliste ist bei solchen Diensten denkbar einfach, genauso wie auch die Einrichtung von moderierten Mailinglisten. Die Gründe für moderierte Mailinglisten können genauso im übernächsten Abschnitt entnommen werden. Gründe für Mailinglisten gegenüber Newsgroups gibt es mehrere, der wichtigste dürfte sein, daß man bei Mailinglisten auch den Empfängerkreis einschränken kann.

Newsgroups und NNTP (kurz)

Im Gegensatz zu dem „routing“ von Emails sind News generell öffentlich zugänglich. Sie sind in einem riesigen Netzwerk verteilt, welches eine an jedem beliebigen Punkt dieses Netzwerks eingelieferte Nachricht an mehrere andere Server weiterleitet, diese leiten weiter usw., bis die Nachricht endgültig auf „allen“ Servern gespeichert ist. Sie ist somit auch von jedem Punkt aus mit Hilfe eines speziellen Clients abrufbar. Dabei ist es unerheblich, welchen Client man benutzt, solange dieser sich an die Standards hält. Das Usenet, wie das Netzwerk auch heißt, ist in viele verschiedene Newsgroups unterteilt. Diese sind im Allgemeinen in Hierarchien und Unterhierarchien unterteilt. Getrennt werden die Ebenen jeweils mit einem Punkt.

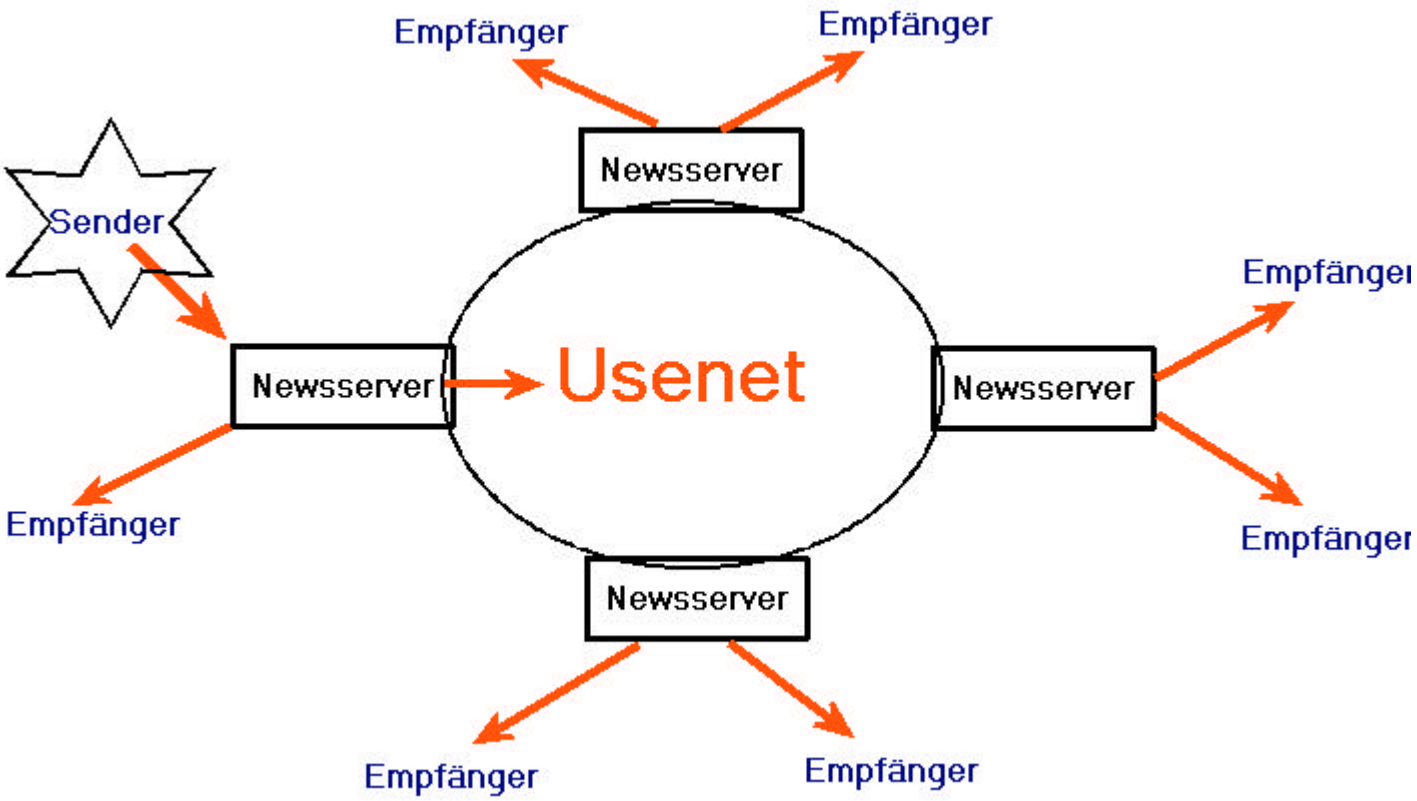
Als Beispiel soll die Newsgroup „hamster.de.talk“ gelten, welche in der Toplevel-Hierarchie „hamster“ gelegen ist. Als Unterhierarchie gilt „de“. In diesem Fall ist dies eine Nominatur für „Deutschsprachig“. „talk“ zeigt nun, daß in dieser Gruppe über Gott und die Welt geredet werden kann. Das Thema der Gruppe kann allgemein durch die komplette Namengebung und Eingliederung in die Hierarchie abgelesen werden. Doch auch dies hat Schranken; in den meisten Hierarchien gibt es daher eine regelmäßig veröffentlichte Nachricht, die alle nötigen Informationen zusammenfaßt. Auch die Regeln der Hierarchie werden regelmäßig veröffentlicht. In diesen stehen die „Benimmregeln“ und Gepflogenheiten, die man berücksichtigen sollte.

Im Ursprung hatte das Usenet kein spezielles Protokoll, sondern bestand aus „öffentlichen“ Emails, welche von einem Computer bzw. von einer Mailbox zur anderen kopiert wurden. Hierzu wurden verschiedene Protokolle benützt, z.B. UUCP.
Auch heutzutage gibt es diese Arten der News-Übermittlung, doch sie ist den „Profis“ bzw. fortgeschritteneren Benutzern vorbehalten, welche die gewählte Art im Normalfall für spezielle Anwendungen brauchen. Die Inanspruchnahme eines solchen Dienstes kostet meistens Geld.
In den 80er Jahren entstand dann ein neues Protokoll eigens für das Usenet namens NNTP („NetNews Transfer Protocol“). Die (Weiter-)Entwicklung dieses Protokolls ist noch nicht abgeschlossen. Für beides gilt als „Referenz“ der Unix-Server INN.
NNTP ist in zwei Bereiche gegliedert:

8.10. Kommunikation zwischen verschiedenen Servern, auch „Peering“ und „Feeden“ genannt. Dieser Bereich wird u.a. durch RFC 977 spezifiziert.

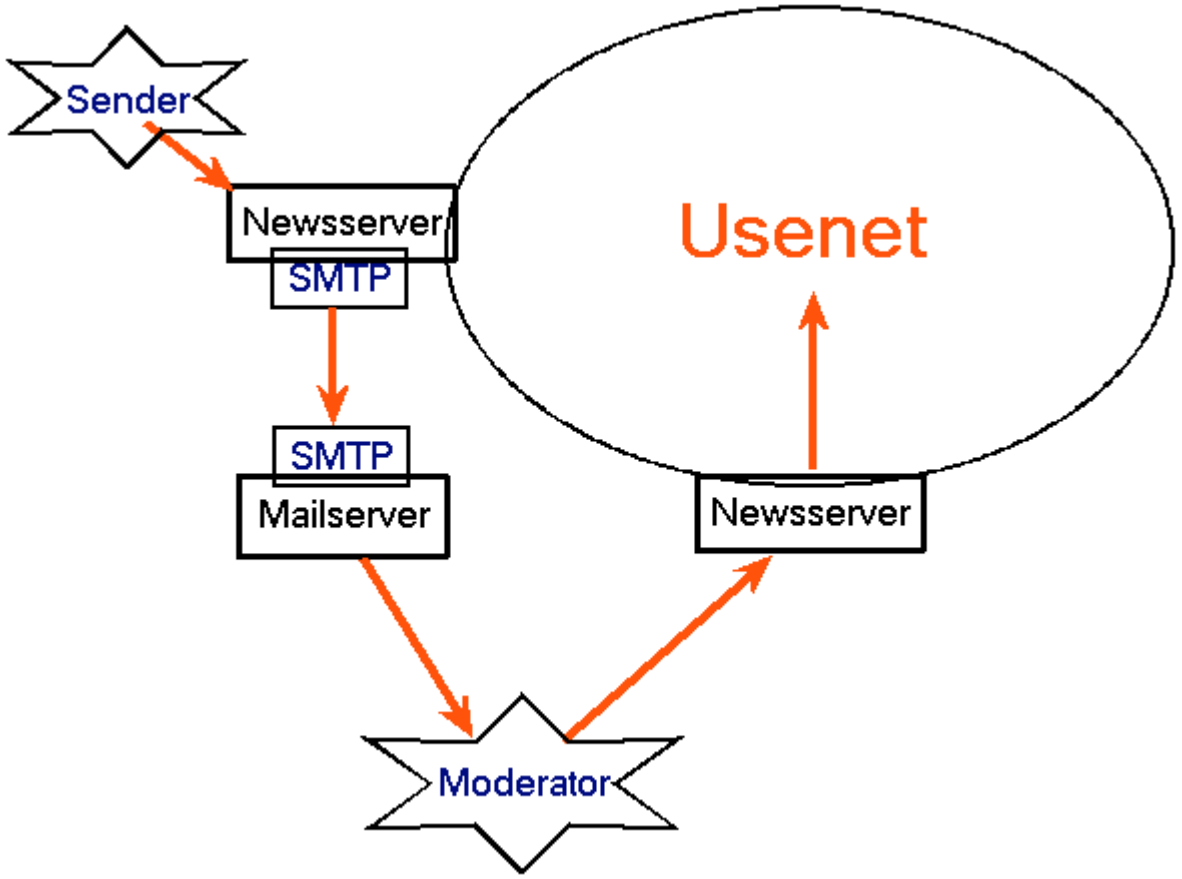
8.11. Kommunikation zwischen Server und Client, auch „NNRP“ („NetNews Reader Protocol“) genannt. Dieses wird u.a. durch RFC 1036 spezifiziert.

„Routing“ von normalen Newsgroups



Der Sender sendet eine Nachricht an irgendeinen Newsserver im Usenet. Dieser routet diese Nachricht über das Usenet zu allen anderen Newsservern. Die Empfänger können hinter jedem dieser Server sitzen, auch dem Einliefernden, um die Nachricht abzurufen. Einzig zu beachten ist, daß nicht jeder Newsserver jede Gruppe führt. Auch gibt es in zugegeben sehr seltenen Fällen „Inseln“, bei denen eine Newsgroup nur auf wenigen oder gar nur einem Server existiert. Oder es existieren mehrere Teilinseln, zwischen denen Nachrichten nicht geroutet werden können, also Diskussionen nur einen Teil der Benutzergruppe erreicht. Auch haben manche Newsserver ‚lokale‘ Newsgroups, d.h. nur Clients, die direkt von diesem Server ihre Nachrichten laden, können diese Newsgroups lesen. Die Nachrichten werden nicht nach außen geroutet.

„Routing“ von moderierten Newsgroups



Der Sender schickt seine Nachricht wie bisher an den Newsserver. Dieser schaut in seiner Datenbank nach, ob diese Gruppe moderiert ist. Wenn sie es denn ist, dann sendet er die Nachricht nicht an das Usenet und stellt sie auch nicht intern zur Verfügung, sondern per SMTP (Email) an die eingestellte Moderatorenadresse. In den meisten Fällen ist dies <newsgroup>@moderators.isc.org. Nun entscheidet der Moderator ob die Mail für die Öffentlichkeit in der Newsgroup gestellt werden soll. Falls dies der Fall ist, sendet er die (u.U. auch veränderte und signierte) Nachricht an irgendeinen(!) Newsserver mit dieser Newsgroup mit einem besonderen Header „Approved“. Dieser Newsserver verteilt dann die Nachricht in das Usenet.

Arbeit einer Moderation

Im Allgemeinen besteht eine Moderation aus mehreren Moderatoren; sie hat eine eigene Emailadresse und meistens einen eigenen PGP-Key. Durch das Moderieren einer Gruppe kann man verhindern, daß frei in diese gepostet wird.
Es gibt verschiedene Gründe, warum eine *Newsgroup* moderiert ist:

a) Announce-Newsgroups sollen möglichst von restlichen Diskussionen befreit sein.
b) Es soll verhindert werden, daß Nachrichten gepostet werden, welche zu weit vom Thema/Gebiet der Newsgroup abweichen.
c) Es soll verhindert werden, daß unbedarft gepostete Informationen, die dem Poster schaden könnten, an die Öffentlichkeit geraten.
d) „Spammer“ (Werbung) sollen es schwerer haben in dieser Gruppe zu posten.
e) Es soll verhindert werden, daß sog. „Fakes“, „Trolls“ oder Poster, welche für ihre Streitlust bekannt sind, das Gruppenklima vergiften.

Technisch[3] ist es für „Wissende“ recht einfach eine Newsgroup zu moderieren, es ist eine Emailadresse vonnöten, an welche die Nachrichten gehen. Dort holt man sie ab und speichert sie auf dem Computer als einfache Textdatei. Nun sollte man, je nach verwendetem Emailprogramm, mit mehr oder weniger vielen Änderungen, eine RFC (2)822-konforme-Nachricht vorliegen haben. Nun werden vom Moderator unnötige Header gelöscht, bestimmte Header verändert wie „Date“, „Message-ID“ und einige andere. Auch müssen einige Header hinzugefügt werden, insbesondere der „Approved“-Header. Als nächstes wird, je nach Anforderung, eine Signatur angehängt und/oder die Nachricht per PGP/ GnuPG signiert. Als letztes kommt nun der Transfer der Nachricht in das Usenet. Dazu ersetzt man jeden ‚Punkt‘ am Anfang jeder Zeile der Nachricht durch einen ‚Doppelpunkt‘, d.h. zwei Punkten hintereinander. Zum Transfer an sich kann man irgendeinen Newsserver, auf dem man Postingrechte hat, per TELNET ansprechen, sich anmelden und dann die Nachricht per POST-Befehl und Copy’and’Paste in das Telnet-Programm posten.
Diese doch recht umfangreiche Arbeit wird heute durch eine Vielzahl von Tools unterstützt. So gibt es Skripte, welche die Änderungen und sogar die Signierung mit Hilfe von PGP auf wenige Tastendrücke oder komplett automatisch vornehmen. Auch gibt es Programme, welche bestimmte Poster direkt „approven“, also bestimmten Benutzern direktes Schreibrecht in die Gruppe gewähren. Dies ist für die Verringerung der Last der Moderatoren sehr nützlich, doch auch diese Tools haben ihre Grenzen.

[3] Eine genauere Beschreibung des Vorgangs: http://www.landfield.com/usenet/moderators/handbook/

Idee zur Verbesserung

Von NNTP zu IMAP

Wie bereits in „Themenwahl ➔Erweiterung der Moderation“ beschrieben hat mich Jürgen Helbing auf diese Idee durch seine Nachricht in news.software.nntp gebracht. Er stellte sich eine Lösung innerhalb von NNTP/Usenet vor, bei der eine moderierte Newsgroup grundsätzlich einen „Home“-Server hat, auf dem man die eintreffenden Nachrichten entweder approven oder löschen könnte. Dazu wären bei gängigen Newsclients gerade einmal zwei Buttons mehr nötig, die bestimmte Aktionen (Befehle) ausführen würden. Beim Senden der Nachricht an eine moderierte Newsgroup würde der Client den Server durch ein DNS ähnliches System ermitteln und direkt an ihn senden. Für diese Lösung wären nicht nur Änderungen an den Clients der Moderationen, sondern auch an den „Home“-Servern nötig. Diese Änderungen wären machbar.
Es gibt aber noch weitere Nachteile, die gegen ein solches System sprechen:

a) Die „Home“-Server bräuchten „anonymous“-Access für moderierte Newsgroups
b) Es wären umfangreiche Änderungen an den einliefernden Clients vonnöten, praktisch also an allen Clients in allen denkbaren (eingesetzten) Versionen. Dieser Nachteil würde dadurch aufgehoben, daß nur die Server, bei denen die Nachrichten eingeliefert würden, die nötigen Änderungen verpaßt bekommen. Doch dabei wäre der entscheidende Vorteil dahin.
c) NNTP ist ein dezentral ausgerichtetes Protokoll. Man kann an jeder Stelle des Netzwerkes die gleichen Nachrichten bekommen.
d) Der Moderator hat nur begrenzte Möglichkeiten das Posting zu verändern. In manchen Fällen (Ausschluß von bestimmten Sendern) ist dies nicht nötig, doch sobald man eine „Announce“-Newsgroup hat, bei der z.B. die Message-ID geändert werden soll, muß man bei dieser Lösung im Server eingreifen. Dies ist insbesondere bei großen Servern nicht möglich.

Diese Methode hat natürlich, vor allem zum bisherigen System, Vorteile:

a) Die Nachrichten bleiben im gleichen Medium. Es sind **keinerlei** Konvertierungen/Workarounds nötig. Der letztgenannte Punkt ist besonders wichtig, da z.B. die Medien Mail und News immer weiter auseinander driften und u.U. vollkommen andere Formate speichern bzw. verteilen. Dazu schreibe ich weiter unten näheres.
b) Bei moderierten Gruppen, die Binaries (Dateien, z.B. ZIP) enthalten, muß der Moderator diese nicht speziell herunterladen, sofern er diese nicht scannen will (z.B. auf Viren).
c) Es gibt keinerlei Probleme mit den „Mailforwardern" aus dem bisherigen System. Jede Nachricht, die an eine moderierte Newsgroup geht, geht über einen dieser Forwarder „*@moderators.isc.org". Diese Forwarder arbeiten bislang kostenlos, doch bei zunehmendem Datenaufkommen, insbesondere durch Binaries, können diese diesen Dienst nicht mehr kostenlos der Netzgemeinde zur Verfügung stellen. Auch auf dieses Problem gehe ich weiter unten im Detail ein.

Im Laufe der (auch hitzigen) Diskussion stellte ich einen Gegenvorschlag zur Diskussion, den ich später noch verbessert habe. In den nachfolgenden Punkten skizziere ich diese Idee:

Das bestehende System wird übernommen. Das heißt: zu moderierende Nachrichten werden über die bisherigen Forwarder an die Emailadresse der Moderation geschickt. Diese ist praktisch ein einziger IMAP-Account auf einem Server, der die Moderations- Erweiterung des IMAP-Protokolls unterstützt. Diese Erweiterung werde ich genauer unter „Implementation" beschreiben. Die Moderation kann nun auf diesen IMAP-Account die Nachrichten mit den Mitteln von IMAP (**auch teilweise**) abrufen. Als nächstes kann der zuständige Moderator diese Nachrichten verändern und diese Veränderungen „uploaden" (auf dem Server ersetzen). Dieses muß in der angesprochenen Erweiterung insofern verbessert werden, daß die Nachricht auch in Teilen verändert werden kann; bislang kann man nur eine komplette Nachricht „uploaden" Als letztes speichert der Moderator ein bestimmtes „Flag" auf dem Server, welches angibt, ob die Nachricht nun zum „approven" bereit ist. Das eigentliche „Approven" findet mit Hilfe eines Befehls ähnlich „EXPUNGE" statt. Dieser löst einen Mechanismus auf dem Server aus, der alle Nachrichten, die dieses „Flag" tragen, per NNTP in das Usenet postet.

Für Moderationen, die einen BOT (Automatismus) einsetzen, z.B. um bestimmte Postern direkt zu „approven“ oder bestimmte Header zu setzen, können diese nun vor dem eigentlichen „approven“ die IMAP-Box durchsuchen lassen. Es gibt keinerlei Probleme mit mehreren Moderatoren in einem Moderationsteam, solange diese nicht an einer einzigen Message gleichzeitig arbeiten. Und selbst das sollte nur bedingt ein Problem darstellen, da man bei IMAP „Responses“ (eine Antwort, eingeleitet durch ein ‚*‘) jederzeit empfangen können muß. Damit kann ein Client auf eine Veränderung der momentan editierten Email/Nachricht reagieren. So z.B. den Moderator warnen. Doch wie kann man dem Server mitteilen, welche Header nun zum „Approven“ behalten werden sollen und welche nur zum anzeigen/mailen benutzt werden sollen? Außerdem gibt es bei Emails andere, teilweise strengere, Regeln für Header. Dieses Problem wurde von den Leuten, die einen der NNTP-RFC erweitern und erneuern wollten, aufgegriffen und daraus folgte der Vorschlag, daß News-Nachrichten als eigenen „MIME“-Typ angesehen werden, der eigentliche Artikel inklusive Header also unabhängig vom Rest der Mail ist. Diese Idee will ich, unabhängig ob sie sich irgendwann im RFC wiederfinden wird, weiter verfolgen.

Eine Entlastung der kostenlos zur Verfügung gestellten Emailforwarder könnte durch einen simplen Trick vonstatten gehen: Anstatt wie bisher alle Nachrichten über einen Account „news.group@moderators.isc.org“ laufen zu lassen, würde dieser grundsätzlich für alle Newsgroups so umbenannt, daß die Hierachie der Newsgroup im DNS-Teil der Mailadresse liegt. Also „news.group@news.moderators.isc.org“.

Als Beispiel sei die bisherige Moderations-Adresse „hamster.de.announce@moderators.isc.org“ genannt, die ggf. „hamster.de.announce@de.hamster.moderators.isc.org“ heißen würde.

Jetzt kann man im DNS-Record für „news.moderators.isc.org“ den „Forwarder 1“ und für „hamster.moderators.isc.org“ „Forwarder 2“ als Mailexchange-Server eintragen. Dadurch kann man eine gute Verteilung der Belastung, insbesondere bei Binary-Gruppen, erreichen.

Um einen letzten Vorteil der IMAP-Lösung gegenüber der NNTP-Lösung deutlich zu machen: Das Moderieren von Mailinglisten ist bei der NNTP-Lösung auch theoretisch nicht möglich, man müßte also die doppelte Arbeit machen, einerseits um NNTP-Nachrichten zu moderieren und Clients für diese Anwendung zu schreiben, andererseits für Mailinglisten und dasselbe mit Emailclients. Man kann durchaus ein Gateway zwischen beiden Medien schreiben, würde dabei aber auf recht komplexe Probleme stoßen. Im IMAP-System müßte man nur dem IMAP-Server beibringen einerseits News zu senden, andererseits Emails. Aufgrund der Verschiedenartigkeit von Mailinglist-Software könnte sich auf Dauer trotzdem eine Art Gateway durchsetzen. Dies hat aber die Zeit zu zeigen, und sollte nicht durch das System an sich beeinflußt werden.

Reichweite

Ich bin mir bewußt, daß die vorgeschlagenen Änderungen enorme Umstellungsarbeiten erfordern würden und daß sich diese nicht unbedingt in allen Produkten umsetzen lassen. Auch ist es sehr unwahrscheinlich, daß all diese Produkte sich komplett an den Standard halten werden. Eine weitere Einschränkung muß man dahingehend machen, daß es bisher nicht sicher ist, ob einer der dargestellten Abläufe auch wirklich die Zustimmung der Mehrheit finden wird. Da der IMAP-Vorschlag im Prinzip keinerlei Änderungen an irgendeinem bestehenden System voraussetzt, sondern auch lokal auf den Rechnern der Moderatoren bzw. auf einem von ihnen „kontrollierten" bzw. dafür vorgesehenen Fremdrechner geschehen kann, sollte der Vorschlag sich leichter durchsetzen lassen.

Vom technischen Aspekt aus gesehen erfordert die IMAP-Lösung doch recht komplexe Clients. Bisher gibt es nur eine recht geringe Anzahl an IMAP-Kompatiblen Clients, die den **bisherigen** Standards genügen. Leider sind von diesen die meisten „Closed-Source", so daß man kaum die Möglichkeit für eine Referenz-Implementation bekommen kann. Dagegen ist das NNTP-Protokoll sehr einfach gestrickt, so daß diesbezüglich weniger Probleme auftreten können.

Serverseitig ist der Vorschlag recht komplex, würde aber, relativ zum bisherigen IMAP-Protokoll gesehen, nur gering mehr Ressourcen brauchen. Die Hauptprobleme sollten die Unterstützung von NNTP zum endgültigen Posten der Nachrichten und die eingangsseitige MIME-Transformation sein.

MIME-Typ „Moderation“

Durch den „Grandson“ des RFC 1036 wäre der MIME-Typ „application/news-transmission“ eingeführt worden. Da dieser Draft nie zu Ende geschrieben wurde, wurde dieses System der Verlustfreien Übertragung der zu moderierenden Nachrichten nie offiziell eingesetzt. Die weiteren Vor- und Nachteile werden genauer im Grandson dargestellt, deshalb sind die relevanten Stellen im Anhang vertreten.

Wir können aber diesen MIME-Typ benutzen um eines **unserer** Probleme zu lösen:

Wie kann man Nachrichten frei von Verlusten speichern, sie dabei in eine „endgültige“ Form bringen und unter Umständen sogar Kommentare für andere Moderatoren hinzufügen? Wir speichern die Nachricht als „application/news-transmission“ (MIME) ab, erzeugen einen Umschlag (Envelopeheader + “multipart/mixed“) und können somit für Kommentare noch weitere MIME-Teile anhängen.

Hierzu sollten wir aber definieren, in welcher Stufe der eigentliche MIME-Typ (news-transmission) angewandt werden sollte. Bereits im Umschlag sollte deutlich werden, daß es sich hier um eine zu moderierende Nachricht handelt. Dazu definiere ich den neuen Header „X-Moderated: Yes“. Wie auch im Teil „Weiterführung in einen RFC“ deutlich werden wird, ist man gezwungen bestimmte Schritte zu unternehmen, so daß der MIME-Typ „registriert“ ist. Aber das soll uns vorerst nicht weiter kümmern.

Als Beispiel für den neuen MIME-Typ bzw. dem zu erzeugenden Umschlag moechte ich diese (Test-)Nachricht liefern:

```
Subject: Test
From: tests@beslernl.goldpool.org
Newsgroups: hamster.modtest
Date: Thu, 10 Apr 2003 10:09:51 +0200
Message-ID: <junk01@beslernl.goldpool.org

Testnachricht zum approven
```

Diese Nachricht geht dann an die Moderation:

```
Subject: Test
From: tests@beslernl.goldpool.org
Newsgroups: hamster.modtest
Date: Thu, 10 Apr 2003 10:10:20 +0200
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
X-MODERATE: YES
Message-ID: <foo@serv.er.invalid
Content-Type: multipart/mixed;
     boundary="------
_3E835F07EA110318A008_MULTIPART_MIXED_"

--------_3E835F07EA110318A008_MULTIPART_MIXED_
Content-Type: text/plain
Content-Transfer-Encoding: 7Bit

Diese Nachricht ist als Test fuer die IMAP-Moderations-
Erweiterung geschrieben worden. Sie ist nicht zu
publizieren!

--------_3E835F07EA110318A008_MULTIPART_MIXED_
Content-Type: application/transmission
Content-Transfer-Encoding: quoted-printable
Subject: Test
From: tests@beslernl.goldpool.org
Newsgroups: hamster.modtest
Date: Thu, 10 Apr 2003 10:09:51 +0200
Message-ID: <junk01@beslernl.goldpool.org

Testnachricht zum approven
--------_3E835F07EA110318A008_MULTIPART_MIXED_
```

(Hierfür benutze ich das Script „moderation.hsc“, das im Anhang zu finden ist.)

Die Moderation bzw. der Server kann hieraus wieder die Originale Nachricht herausbekommen, und zwar **ohne** Verluste.

Was soll geleistet werden

Protokoll

Das Hauptproblem beim schreiben eines Protokolls bzw. der „Erweiterung“ eben dieses, besteht darin, daß man einerseits die bestehenden RFC ausnutzt, gleichzeitig aber keinen verletzt. Insbesondere RFC 2060 (bzw. neu: RFC 3501) hat einige Verbote definiert, an denen man einige Erweiterungen anknüpfen könnte. Außerdem ist, wie bereits erwähnt, die ‚Szene‘ noch stark in der Entwicklung. Der RFC 3501 ist ‚taufrisch‘ (Ende März 2003) herausgekommen, so daß einige der dort neu hinzugekommenen Gebiete/Klarstellungen zum RFC2060 (ehemaliger IMAP-Standard) in diesem Dokument nur schwer eingehalten/berücksichtigt werden konnten.
Ob ich in der „Besonderen Lernleistung“ unbedingt RFC-treu sein kann, mag ich bezweifeln. In dem Teil „Weiterführung in Form eines RFC“ werde ich auf die genaueren Probleme und Unklarheiten eingehen, soweit sie für mich abschätzbar sind. Viele Dinge werden sicherlich von Mark Crispin, dem Autor des RFC 2060/3501, anders gesehen oder auch sonst auffallen, so daß dieser bei der Weiterführung zum RFC sicherlich noch einige Änderungen und Verbesserungen einbringen wird.

In den folgenden Punkten sind die benötigten Erweiterungen am IMAP-Protokoll zusammengefaßt:

a) Ein Ordner muß als „Moderiert“ markiert werden. Im Allgemeinen ist hierbei der Name des Ordners unwichtig, da dem Accountinhaber bekannt ist, daß dieser Ordner für Moderation gedacht ist. Trotzdem muß man dies der Client-Software mitteilen.
b) Es müssen die moderierten Nachrichten als „Application/ news-transmission“ vorliegen. (Egal ob dieser Umschlag nun serverseitig oder clientseitig erzeugt wurde)
c) Dem Server muß gesagt werden können, daß eine Nachricht (bzw. der news-transmission-Teil) per NNTP gesendet werden kann.
d) Man muß den (MIME-)Teil ändern können. (z.B. Änderung des Nachrichtenheader)
e) Ein Kommentar muß eingefügt werden können (u.U. gleich Punkt d)

Aus diesen Punkten kann man zwei Bereich ableiten:

a) Erweiterungen des IMAP-Protokolls zum *Verändern* der (MIME-)Daten
b) Erweiterungen zum moderieren/posten

Im Rahmen der „Besonderen Lernleistung“ werde ich nachfolgend nur b) behandeln, ersterer Punkt würde mächtige, neue Befehle erfordern. Außerdem kann man diese durch ‚Append‘ simulieren, indem man die alte Nachricht komplett lädt, dann löscht und danach per ‚Append‘ wieder hochlädt. Dies würde aber die Gefahr durch gleichzeitigen Zugriff von mehreren Moderatoren erhöhen.
Die Erweiterung der Moderation hingegen ist eine erreichbare Aufgabe. Von den vorherigen geht dieser 'Überpunkt' aus den Punkten a, b und c hervor, wobei b) zu beiden Teilen paßt.
Zu a) kann man sagen, daß es bereits im bisherigen Protokoll beim „LIST“ (Anzeigen der Ordnerstruktur) für jeden Ordner gewisse „Flags“ gibt. Standardmäßig wird z.B. „Unselectable“ unterstützt. An dieser Stelle kann man einen „Flag“ hinzufügen, das dem Emailprogramm mitteilt, daß dieser Ordner zum Moderieren benützt wird.
Zu b) muß man sich entscheiden, ob der Client oder der Server die Nachrichten in das entsprechende Format bringen soll. Der beste Weg ist wohl dem Server diese Aufgabe zukommen zu lassen, da unterschiedliche Clients u.U. verschiedene Bugs bei der Erstellung/Behandlung haben können, die im Client für sich alleine nicht stören, gemeinsam aber eine Moderation lahmlegen könnten. Außerdem fällt damit auch der lästige Schönheitsfehler weg, der zum Laden der kompletten Nachricht zwingt. Auch ist anders das Ändern von Teilnachrichten nicht möglich. Dies würde ein entscheidender Vorteil gegenüber dem bisherigen System darstellen. (siehe oben)
➔Beim einsortieren von eingehenden Emails zum Moderator-Account **muß** die Umsetzung zum „Application/news-transmission” durch den Server geschehen.
Zu c) kann man wiederum, zu mindestens einem Teil, auf bestehende Befehle aufsetzen. Man kann bisher jeder Message bestimmte „Flags” zuordnen, also z.B. „\Seen” oder „\Answered”. Darauf aufbauend kann man einen neuen Flags einführen, der markiert ob eine Nachricht gesendet werden kann. Dazu würde ich den Flag „Approved“ vorschlagen. Dies hat zudem den Vorteil, daß die IMAP-RFC grundsätzlich eine Erweiterung dieser Flags zulassen.
Jetzt benötigt man noch einen extra Befehl, der diese Flags „ausführt“, d.h. Nachrichten endgültig in das NNTP-System einführt. Man kann zwar den Befehl „EXPUNGE“ benutzen, welcher dazu da ist um gelöschte Nachrichten endgültig vom Server zu entfernen (vorher werden sie nur als gelöscht markiert). Doch da dieser schon seine definierte Handlung hat, **muß** man einen neuen Befehl einführen. Aus meiner Sicht ist der beste Name dafür: „XAPPROVE“

Wir müssen nun entscheiden, was und ob der Server nach diesem Befehl zur Nachricht hinzufügt, man kann es durchaus auch offenlassen, so daß ‚lokale' Server automatisch einen Approved-Header setzen **dürfen** o.ä.
➔Wenn ein Server einen Ordner als „moderiert" angibt, **muß** er dort auch den Flag „Approved" zulassen. Außerdem **muß** er den Befehl „APPROVE" kennen und interpretieren.
➔Ein solcher Server (zum moderieren von News-Nachrichten) **muß** Zugang zum Usenet haben
Um dem RFC 2060/3501 bzw. dem Standard für Erweiterungen gerecht zu werden, hat ein Server, sofern er diese Erweiterung unterstützt das Token „XMODERATION" in der Antwort auf den Befehl „CAPABILITY" zu senden. Das hat zur Folge, daß der Client davon ausgehen kann, daß der Server ihm die Möglichkeiten, die in diesem Teil besprochen wurden, bietet.

Server

Serverseitig muß man recht viele Änderungen beachten. Die Änderung die je nach Speicherart wohl am einfachsten zu implementieren sein sollte, dürfte das Konvertieren in den MIME-Typ „news-transmission" sein. Hierbei muß nur ein MIME-„Umschlag" beim ändern der Emails erzeugt werden. Das eigentliche Schwierige ist der MIME-Typ selber, der z.B. Kommentare zulassen sollte aber gleichzeitig zur bisherigen MIME-Unterstützung/Praxis kompatibel ist. Hierfür ist später eine **strikte** Standardisierung nötig, so daß Mißverständnisse gar nicht erst auftreten können.
Andere Probleme bereiten andere Teile der Serverimplementation, sofern z.B. die Möglichkeit neue Flags für Nachrichten/Ordner einzubauen, zumindestens intern noch nicht vorgesehen ist. Allein schon die Umsetzung dieser Neuerungen würde Programmierarbeit bedeuten.
Der weitaus wichtigste und komplexeste Teil der Umsetzung auf Seite des Servers ist das „Approven" an sich, d.h. das einliefern der Nachrichten in das Newssystem. Hierbei ist vor allem zu beachten, daß in manchen Umgebungen es sinnvoll sein kann den Approved-Header vom Server setzen zu lassen, in anderen variiert er aber von Moderator zu Moderator. Auch hier ist der Standard (RFC) gefordert hart durchzugreifen und nur eines der beiden zuzulassen, oder um zu einer Lösung zu tendieren.
Hierbei gilt auch zu beachten, daß der Vorteil, daß der Client seinen ‚eigenen' Header hineinschreibt und somit konfigurierbar für den Anwender macht, sich nachteilig für die Programmierung der Clients auswirkt. Die Änderungen dort werden dadurch noch komplexer.

Client

Clientseitig sind nur recht wenige Änderungen vonnöten, im normalen Client, welcher den RFC 2060 (bzw. 3501) vollkommen unterstützt, muß man insbesondere diese Dinge beachten:

a) Man braucht eine Möglichkeit die Flags „Approved“ (und „Declined“) zu setzen. Dafür sind zwei neue Buttons z.B. „Setze Approved-Flag“ nötig, oder gar nur eine Erweiterung des Kontextmenüs (Pull-Down-Menü)
b) Der MIME-Typ für moderierte Nachrichten muß unterstützt werden. Für die gängige Software sollte dies kaum ein Problem darstellen, außer in Kombination mit nachfolgendem:
c) Die Änderung von Teilen der Nachricht muß eingebaut werden, bei technischer Unmöglichkeit muß es eine Möglichkeit geben, die alte Nachricht zu löschen und die neue Nachricht hochzuladen. (Siehe Abschnitt „Protokoll“)
d) Der Befehl APPROVE sollte auslösbar sein. (Toolbutton/Menü)
e) Es **sollte** eine Möglichkeit geschaffen werden einen BOT anzusprechen oder gar per Script zu schreiben.
f) Man muß mit mehreren Clients zugleich die gleiche Mailbox Online(!) lesen können. Der Client muß dafür ggf. ausgelegt sein und entsprechende RFC eingehend unterstützen. (Siehe u.a. RFC2177)
g) Bei Implementierung/Konzeption muß eine Unterstützung für „Moderationsordner“ geschaffen werden. Diese dürfen aber für andere Clients keine Probleme ergeben.

Probleme

Die meisten Probleme sollten bereits in den oberen Punkten deutlich geworden sein, daher sehe ich hier nur eine Liste der wichtigsten Probleme vor, vor allem derjenigen die auch weiterhin bestehen, auch nach dieser „Besonderen Lernleistung“. Sie sollten in der eigentlichen Standardisierung (RFC) angesprochen oder definiert werden.

a) Das später benötigte Ändern von *Teilen* einer Nachricht
b) Ordner als „Moderiert“ zu deklarieren (kompatibel zum RFC!)
c) MIME-Typ genau festzulegen ohne Einschränkungen für Erweiterungen machen zu müssen
d) Der Server muß Zugang zum Usenet haben (Bei Hamster als zusammenhängender Mail-/Newsserver kein direktes Problem)

Implementation

Allgemeines

Leider hatte ich bei der Implementation nur die Möglichkeit Hamster zu erweitern, da ich keinen IMAP-Clienten kenne, bei dem es in angemessener Zeit möglich wäre, die Erweiterung einzubauen und dessen Source außerdem noch frei verfügbar ist. Den Versuch einen eigenen IMAP-Clienten basierend auf einer TCP/IP-Komponente zu erstellen, habe ich nach wenigen Tagen verworfen, da dies nicht unter dem Begriff „angemessene Zeit“ fällt. Vielleicht werde ich dieses recht interessante Projekt irgendwann anders weiterführen. Das Erstellen eines IMAP-Clienten, der möglichst konform zu den RFC ist, ist nunmal sehr kompliziert. Fast sogar komplizierter als das reine Implementieren des Servers. Zum testen der serverseitigen Erweiterung im Hamster habe ich daher das Programm/Tool „Telnet“, das in jeder Windows-Installation existiert, benutzt.
Je nach Anforderung kann man mit Telnet jedes Protokoll ‚simulieren‘, sofern es zeichenbasiert ist. Doch hierzu werde ich später weiteres sagen, wenn es an das Testen von Hamster geht. Vor dem Testen sollte zunächst die Implementation des Servers gemacht werden...

Änderungen im Hamster

Ich beziehe mich bei diesen Änderungen ausschließlich auf meine veränderte Hamsterversion 2.0.1.30001 „Enhanced“, spätere Versionen, insbesondere die der „Classic“-Linie können meine Änderungen beinhalten, müssen es aber nicht. Außerdem kann die Erweiterung infolge von Weiterführungen in ihren Bestandteilen geändert werden (müssen). Alles in allem muß man recht vorsichtig mit der Bewertung des Quellcodes im Hamster sein.
Folgende Änderungen wurden vorgenommen (Source ist im Anhang):

1) Ich habe für die IMAP-Erweiterung eine Art „Newsrouter“ für ausgehende Nachrichten geschrieben. Über diesen gehen alle am NNTP-Server eingelieferten Artikel, außerdem die per IMAP-Approve zu sendenden. Dieser „Newsrouter“ war recht komplex und würde bei Komplettierung (Client-NNTP) genauso gut als Thema für eine „Besondere Lernleistung“ dienen können.
2) In cIMAPMailbox.pas und cServerIMAP.pas habe ich die wichtigsten Teile für IMAP-„Approve“ schreiben müssen.
3) Es kamen fünf neue „Actions“ hinzu, d.h. an bestimmten Stellen können Programme gestartet werden
4) In cIMAPMessage.pas kam die Methode „GetModerationPart“ hinzu.

Die Testumgebung (Server)

Installation der Erweiterung im Hamster: Da ich eine ‚fertige' Installation mitliefere, ist für den Tester nichts weiter einzurichten oder zu konfigurieren. Ich beschränke mich daher auf eine ‚simple' Beschreibung der Vorgänge, die vor allem zusätzlich zur normalen IMAP-Installation des Hamsters zu machen wären. Die normale Installation ist in der IMAP-FAQ beschrieben, die in der Hamsterhilfe zu finden ist.

a) Es muß eine Datei „MODERATE" in dem Verzeichnis generiert werden, das später als „Moderations"-Verzeichnis dienen soll.

b) Im Mailfilter (Datei MailFilt.hst) müssen die entsprechenden Emails, die zum Account der Moderation gehen, an das Moderationsverzeichnis des IMAP-Accounts weitergeleitet werden. (Bsp: „Set(IMAP/hamster.de.announce) To: „hamster-de-announce@snafu.de")
Achtung: Diese Datei wird nur beim Abruf per POP3 beachtet. In der Testsuite ist das direkte Senden durch SMTP nachgebildet, daher ist dort die INBOX des Moderators als MODERATED deklariert.

c) Eine Aktion muß für das Eintreffen von Emails dieses Accounts definiert werden. In dem definierten Skript/Programm muß der MIME-„Umschlag" erzeugt werden. Für diese Aktion bietet sich „mail.preprocess" an.
In der Testsuite ist dieser Schritt nicht unternommen worden, siehe oberes Problem.
Der MIME-„Umschlag" wird im „Server" unternommen.

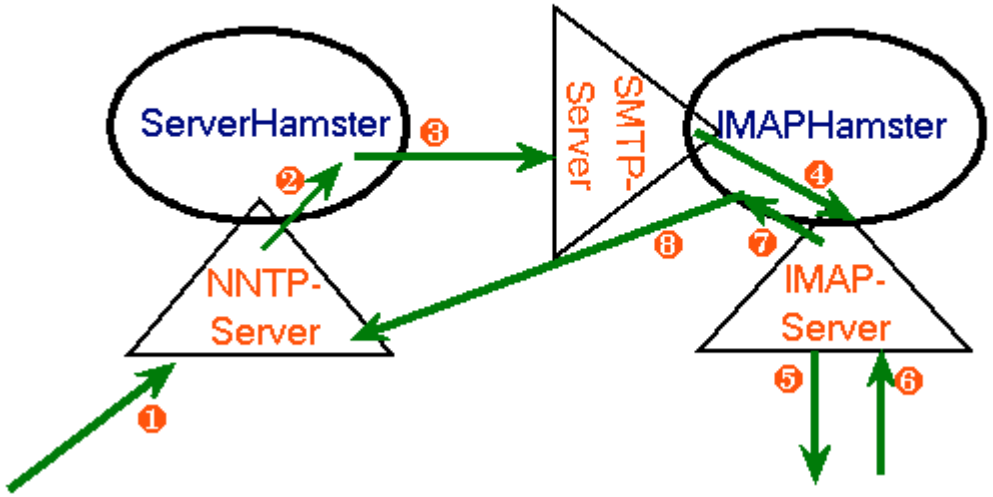
d) Diese Moderations-Erweiterung wird in späteren Versionen nur ‚optional' sein und wird somit in der Hamster.ini angeschaltet werden müssen. Im momentanen Source ist dies noch nicht vorgesehen.

Die Testumgebung (Client)

Da ich keinen Opensource-IMAP-Client kenne, der in Delphi geschrieben ist, hatte ich kurzzeitig versucht diesen selber zu schreiben. Nach anfänglichen Erfolgen habe ich aus Zeitmangel dieses Vorhaben aufgegeben und versucht einen Programmierer eines „kommerziellen" IMAP-Clienten zu finden, der mich in diesem Vorhaben durch das Implementieren im Client unterstützt. Leider haben meine Suchbemühungen keine Erfolge getragen, so daß ich wohl gezwungen bin den „schweren" Weg über „Telnet" zu gehen. Ich werde dementsprechend Logfiles mitliefern, mit denen man vergleichen kann.

Achtung: Wenn man sich vertippt hat, kann man **nicht** per Backspace diesen Fehler berichtigen!

Die mitgelieferte Testsuite



(1) User sendet etwas an den Usenet-Server

(2) Action „n2m.moderate“ wird aufgerufen, somit auch „moderation.hsc“ (Der MIME-Umschlag wird erzeugt)

(3) Der Server sendet per SMTP an das zuständige Moderations-Postfach

(4) Im Mailserver wird die Mail an den richtigen Empfänger zugestellt. Hier könnte eine erneute Prüfung erfolgen, ob die Nachricht mit dem MIME-Umschlag versehen ist. Bisher wird für IMAP-Postfächer keine Action ausgeführt, daher ist die Überprüfung in der Testsuite nicht eingebaut.

(5) Der Moderator kann per IMAP die Nachricht oder Teile davon lesen

(6) Der Moderator verändert die Nachricht (APPEND), fügt das Flag „APPROVE“ an und zum Schluß führt er den Befehl „APPROVE“ aus.

(7) Zuerst wird der MIME-Umschlag entfernt, dann wird hier die Action „imap.approvemod“ ausgeführt und somit „approve.hsc“. In diesem Script werden in der Testsuite die nötigen Header erzeugt.

(8) Der IMAP-Server sendet die Nachricht an den Usenet-Server. In der Testsuite wird im gleichen Zuge die Nachricht auch gepullt.

Die Testsuite soll möglichst große Teile der im Usenet antreffbaren Konfigurationen für Moderatoren abdecken, dabei aber nicht zu kompliziert werden. Ich habe diese daher basierend auf zwei Hamsterinstallationen erstellt. Hamster1 soll einen x-beliebigen Usenet-Server repräsentieren, bei dem die Gruppe auf „moderated“ steht. Dieser Server erzeugt den MIME-Umschlag und sendet die Nachricht per SMTP an den (IMAP-)Mailserver des Moderators (Hamster2). Außerdem stellt Hamster1 das Usenet dar, in welches Hamster2 später die Nachricht approved schickt.
Hamster2 stellt den Mailserver des Moderators dar. Dort kann dieser schalten und walten so wie er will. In dieser Phase des Protokolls, ohne der Erweiterung zur Änderung von Teil-Nachrichten, ist es recht mühselig alles per APPEND und Telnet zu machen. Daher werden in der Testsuite die Nachrichten **nach** dem APPROVE des Moderators mit den zugehörigen Headern (Approved/Sender) per Action ausgestattet.

Der Testablauf

Als erstes erzeugen wir eine Textdatei (nicht Word o.ä., sondern Plain-Text). In diese schreiben wir die Testnachricht. In unserem Fall nehmen wir die selbe Testnachricht, wie wir sie schon für den MIME-Typ benutzten. Achtung: Falls ein Punkt am Anfang der Zeile steht, muß dieser durch zwei Punkte ersetzt werden, d.h. einfach ein weiterer Punkt vorgesetzt werden. Nun starten wir zuerst beide Hamster, dann Telnet und verbinden uns nun mit dem Host 127.0.0.1 und Port 5119. Hamster wird uns mit „*200 NNTP-Server Enhanced Hamster Vr. 2.0 (Build 2.0.1.30001) (post ok) on hamster1.beslernltest.goldpool.org says: Hi!*“ begrüßen. Nun können wir Befehle eingeben und Hamster wird uns jeweils antworten. Jeder Befehl muß durch ein CRLF abgeschlossen werden, d.h. man muß am Ende „Enter“ drücken.
Zuallererst müssen wir Hamster sagen, wer wir sind (d.h. uns authentifizieren). Dies geschieht durch das Eingeben von „*AUTHINFO USER user*“ und nachfolgend „*AUTHINFO PASS user*“. Hamster sollte uns freudig mit „*281 Authentication accepted*“ empfangen. Nun sagen wir ihm, daß wir für ihn eine Nachricht haben: „*POST*“. Nun erwartet er von uns die Nachricht, die wir durch Markieren und Kopieren der vorher angelegten Textdatei und nachfolgend durch das Einfügen in Telnet (Nicht STRG+V) bewerkstelligen. Zum Schluß geben wir das Nachricht-Ende Zeichen an, d.h. ein CRLF, ein „.“ (Punkt) und abschließend wieder ein CRLF. Artig, wie wir sind, melden wir uns ordnungsgemäß per „*QUIT*“ bei Hamster ab.

Jetzt ist der Artikel auf Reisen, vorerst zwischengespeichert im Mail.Out-Verzeichnis des Serverhamsters. Für uns heißt dies, daß wir im Serverhamster den Menüpunkt „Online→SMTP-Server→127.0.0.1,5025" ausführen müssen. Dadurch wird der Artikel an den IMAP-Server geschickt und dort an unseren IMAP-Benutzer geleitet.
Mit diesem melden wir uns nun am IMAP-Hamster an. Der IMAP-Hamster hat durch den Start ohne SSL im Logfile eine für uns vollkommen unerhebliche Warnung. Wir starten wieder Telnet und verbinden uns mit 127.0.0.1 und nun Port 5143. Diesmal begrüßt uns Hamster mit einem lächelndem "* *OK IMAP4rev1 Server Enhanced Hamster Vr. 2.0 (Build 2.0.1.30001) on hamster2.beslernl.goldpool.org greets you!*". Wir authentifizieren uns diesmal mit dem Befehl "*001 LOGIN moderator moderator*". Als nächstes müßen wir die richtige Mailbox selektieren, d.h. in unserem Fall "*002 SELECT INBOX*". Hamster wird uns freundlich sagen, daß "1 EXISTS", d.h. es gibt eine einzige Mail in diesem Postfach. Nun müßen wir das Flag APPROVE der Mail zuordnen, so daß es nachfolgend gesendet wird. Dazu geben wir den Befehl "*003 STORE 1 +FLAGS (\approve)*" an. Als letztes soll dann der Approve ausgeführt werden, dafür ist der Befehl "*004 APPROVE*" da. Als letztes melden wir uns wieder ab: "*005 LOGOUT*"
Nun sollte im News.Out-Verzeichnis des IMAP-Hamsters die Nachricht mit Approved-Header vorhanden sein. Diese senden wir als letztes endgültig zurück in das Usenet, indem wir im IMAP-Hamster den Menüpunkt „Online→NNTP-Server→127.0.0.1,5119" ansprechen. Die Nachricht wird selber, nach dem Approven, als "SEEN" und "DELETED" markiert. Vor allem dieses Verhalten wird sich wohl in folgenden Entwürfen ändern...

Weiterführung in Form eines RFC

Hier folge ich vor allem der Anleitung, die auf www.rfc-editor.org steht, außerdem ist der Standardisierprozess im RFC 2026 festgeschrieben worden. Auch andere Dinge sind auf dem Weg zum Endziel RFC zu beachten: Erstens die Unterstützung bzw. der Wohlwollen von Mark Crispin, der Schreiber der IMAP-RFC. Zweitens die Unterstützung durch NNTP-Serverprogrammierer und –Betreiber. Drittens eine offiziell gültige Referenz-Implementation, am Besten in der Client/Serverkombination, die sowieso als Referenz für IMAP gilt: UW/PINE.
Zuerst muß festgestellt werden, ob der bisher vorliegende „Draft" gegen irgendeinen RFC verstößt, vor allem also RFC 3501. Als nächstes muß ein offizieller „Internet-Draft" formuliert und eingereicht werden. Sobald dieser fertig formuliert ist, und zu einem RFC werden kann, muß der MIME-Typ endgültig registriert werden, wie es die RFC 2045-2048 fordern. Nun sollte es eine letzte „Abstimmung" in den NNTP-Gruppen geben, so daß der RFC auch angenommen würde.

Fazit

Hier könnte ich verschiedene Dinge ansprechen, die mir im Laufe dieser „Besonderen Lernleistung“ auffielen oder mir begegneten. Doch bis auf die Tatsache, daß ich im Verlauf viel gelernt habe, insbesondere MIME habe ich jetzt endlich komplett durchdenken können und verstanden und daß ich auf die Hilfsbereitschaft mir teilweise fast Unbekannter Menschen stieß, denen ich hiermit noch einmal danken möchte, ist die „Besondere Lernleistung“ für mich ein ausschlaggebender Grund gewesen, einige Projekte im Hamster-Source zu realisieren. So ist der Newsrouter komplett dieses Dokuments wegen entstanden, aber auch im IMAP-Teil habe ich einige Verbesserungen und Erweiterungen eingebracht, für die ich ohne der „Besonderen Lernleistung“ entweder zu geringe Bedeutung zugedacht hätte oder aber für die ich durch andere Projekte zuwenig Zeit gehabt hätte.

Weiterhin möchte ich den Lerneffekt betonen, der durch meine kurzen Versuche einen IMAP-Clienten selber zu programmieren ausgelöst wurde. Es ist doch eine vollkommen andere Sache, auf welcher Seite der Programmierung man sitzt.

Diese „Besondere Lernleistung“ dürfte in der Moderationswelt helfen, sobald es vernünftige Clients gibt, die diese Erweiterung unterstützen. Die Implementierung serverseitig im Hamster dürfte als Erfolg gewertet werden können, wie man auch an der Testsuite sehen kann.

Weiterführen kann man diese Arbeit durch die angesprochene Erweiterung des IMAP-Protokolls durch „teilweises Ändern“. Auch das Schreiben eines IMAP-Clients mit dieser Erweiterung kann man als „Erweiterung“ auffassen. Einen RFC hieraus zu formulieren wird das eigentliche Ziel sein, dies wird sich aber zwangsläufig ergeben, wenn man die vorgegebenen Strukturen nur Konsequent weiterführt.

Selbständigkeitserklärung: Ich habe diese Arbeit (Besondere Lernleistung), zusammen mit der Implementation, selbständig erstellt und verfaßt.

Rheine, Freitag den 11.4.2003: unterzeichnet: **Heiko Studt**

Anhang

Nachricht mit Message-ID: b0jelm.qs.1@Heiko.Studt.dialin.t-online.de

```
There was a part in this Thread talking about IMAP and news2IMAP.
Why don't you use Emails and a IMAP-Server at the end of this.
Now there we need an extension of the IMAP-Protocol, like:

"This Folder is a usenet-moderator-folder."
(For example as Option (command+) in LIST)

-----
Server implementing this extension SHOULD (near MUST) have the possibility to
accept more than one connection on the same IMAP-folder.
Comment to next paragraph:
"ModeratorID" can be an email-address, but AFAIR the namespace-extension could
do this too. (I didn't read that RFC completely yet)
-----

For those mod-folders there are three new commands needed:
(LIST have only to be extended)


--------
1. APPROVE <msgset>

Response: Untagged "APPROVED <msgnum> <moderatorID>"
          MUST be sended if _another_ moderatorID connects,
          MAY be purged after x Days (x>7)

    OK – DONE

The message was accepted. It will be sended to a newsserver/the
newsserver-modul.



2. DECLINE <msgset> [Literal Msg]
Response: Untagged "DECLINE <msgnum> <moderatorID>"

    OK – DONE

The Message wasn't accepted. If given, the literal is a response-
Message with comment for the author of the posting



3. APPENDHEADER <msgnum> <Option> [Params as Literal(s)]
Response: "HEADERCHANGE <msgnum> <moderatorID>"
          MUST be sended if _another_ moderatorID connects,
          MAY be purged after x Days (x>7)
   OK -- Header changed
   NO -- Msg not available

Options:
  -ADD
    Like above, but the HeaderName will be added at (bottom|top?) to the
    Header of the message.
  –REPLACE
```

```
    Every param is a literal in the form: {23}HeaderName: HeaderValue
    The HeaderName in the Message will be replaced to the new HeaderValue
  -RENAME
    Every param is a literal in the form {27}OldeaderName:NewHeaderName:
    The first appeareance of OldHeaderName will be renamed to NewHeaderName
    (top/bottom?) Perhaps it's better to make it ALL/TOP/BOT like with DELETE?
  -DELETEALL
    Every param is a literal in the form: {11}HeaderName:
    The HeaderName will be completely deleted in the message. (all)
  -DELETETOP
    Same as above, but only the first appearance from top is deleted
  -DELETEBOT(tom?)
    Same as above, but only the first appearance from bottom is deleted.
  –WHOLE
    There can be only ONE param (literal). The whole header of the message is
    replaced.
--------

Now there is need for clients like:
1. Reading BODY.STRUCTURE or BODY.HEADER.
   Theoretically it can also load some lines of the text-part(s) if
   there are some available
2. Showing it to moderator, listening to APPROVED/DECLINE/HEADERCHANGE
3. Moderator can change headers of the article.
   (Perhaps in the extension there should be a field for comments, deleted
   before sending)
4. Moderator choose Approve/Decline and it's sent.

While Moderator changes header of one article, another moderator could
*perhaps* change the header too. If this happens (HEADERCHANGE is sent)
the client MUST warn the moderator and let him decide how to go further.
(reread of the header, ...)

> If moderators are faced with an elegant, extremely reliable and
> very very easy system then we'll see what's happening.....

The easy use depends *only* on the client of the moderator. Everything else
does relatively not matter. In practical there are many bots available to help
moderators.
```

Mail an Juergen Helbing zur weiteren Idee

```
I thought about our problem a bit again and now I found a simplier solution in
IMAP: You can store flags in IMAP for every single message, so you can also
store a Flag for '\APPROVED' (and later 'DECLINED').

Headers can be read completeness, so we need only (maximum) two new commands:
(Quick and dirty written now, cause I have no time)

MODERATE (APPROVE|DECLINE) (MsgSet)

Execute the appr/decl-Flags


CHANGEHEADER (FIRST|ALL|LAST|APPEND|DELETE) (?MsgSet?) ("Header" "NewContent)
                                                                    List
Changes the headers of the message. Instead of MsgSet we can define to use
_only_ one Message-Number

(Should be done with Append later on)


Both commands can be used also with UID.

UID MODERATE
UID CHANGEHEADER


>   -WHOLE
>     There can be only ONE param (literal). The whole header of the message is
>     replaced.

Perhaps should be done too.
```

Identification of messages delivered via both mail and news (Draft-only)

```
It is also recognised that certain elements of the transport (including,
but not limited to, mail-news gateways, mailing list reflectors, and
newsgroup or mailing list moderators) might modify existing message
bodies and headers.  The modification might be in form only, such as the
Content-Transfer-Encoding ([MIME]) of the body being changed; or it
might be substantive, such as a standard disclaimer, or standard set of
instructions, being appended to the bodies.  This means that software
conforming to this document cannot guarantee that the two messages will
have identical bodies by the time they reach their destinations.  It
can, however, hold that as a goal, with the recognition that the goal
will not always be reachable due to forces beyond its control.  (In
other words, the author believes that transport and gateway software
that so alters the message bodies is wrong in so doing; but recognises
that such software, while in the minority, is not going to go away any
time soon.)
```

RFC 1036

```
.2.11.  Approved
    This line is required for any message posted to a moderated
    newsgroup.  It should be added by the moderator and consist of his
    mail address.  It is also required with certain control messages.
```

Son of RFC 1036

6.10. Approved

The Approved header content indicates the mailing addresses (and possibly the full names) of the persons or entities approving the article for posting:

Approved-content = From-content *("," [space] From-content)

An Approved header is required in all postings to moderated newsgroups; the presence or absence of this header allows a posting agent to distinguish between articles posted by the moderator (which are normal articles to be posted normally) and attempted contributions by others (which should be mailed to the moderator for approval). An Approved header is also required in certain control messages, to reduce the probability of accidental posting of same; see the relevant parts of section 7.

NOTE: There is, at present, no way to authenticate Approved headers to ensure that the claimed approval really was bestowed. Nor is there an established mechanism for even maintaining a list of legitimate approvers (such a list would quickly become out of date if it had to be maintained by hand). Such mechanisms, presumably relying on cryptographic authentication, would be a worthwhile extension to this Draft, and experimental work in this area is encouraged. (The problem is harder than it sounds because news is used on many systems which do not have real-time access to key servers.)

NOTE: Relayer implementors, please note well: it is the POSTING AGENT that is authorized to distinguish between moderator postings and attempted contributions, and to mail the latter to the moderator. As discussed in section 9.1, relayers MUST not, repeat MUST not, send such mail; on

receipt of an unApproved article in a moderated newsgroup, they should discard the article, NOT transform it into a mail message (except perhaps to a local administrator).

NOTE: RFC 1036 restricted Approved to a single From-content. However, multiple moderation is no longer rare, and multi-moderator Approved headers are already in use.

Grandson of RFC 1036 (Draft-Only)

6.10. Approved

The Approved header content indicates the mailing addresses (and possibly the full names) of the persons or entities approving the article for posting:

Approved-content = From-content

An Approved header is required in all postings to moderated newsgroups. If this header is not present then relaying and serving agents MUST reject the article.

An Approved header is also required in certain control messages, to reduce the probability of accidental posting of same; see the relevant parts of section 6.6.

Please see section 7.1 on how injecting agents should treat posts to moderated groups that do not contain this header.

6.15.5.1 Application/news-transmission

The Content-Type "application/news-transmission" is intended for the encapsulation of complete news articles where the intention is that the recipient should then inject them into Netnews. This Application type SHOULD be used when mailing articles to moderators and to mail-to-news gateways.

[The remarks about sending articles to moderators should perhaps be made in a more appropriate place in our standard.]

NOTE: The benefit of such encapsulation is that it removes possible conflict between news and email headers and it provides a convenient way of "tunnelling" a news article through a transport medium that does not support 8bit characters.

The MIME content type definition of "application/news-transmission" is:

MIME type name:	application
MIME subtype name:	news-transmission
Required parameters:	none
Optional parameters:	moderate; inject; relay
Encoding considerations:	A transfer-encoding (such as Quoted-Printable or Base64) different from that of the article transmitted MAY be supplied (perhaps en route) to ensure correct transmission over some 7bit transport medium.
Security considerations:	A news article may be a "control message", which could have effects on the recipient host's system beyond just storage of the article. However, such control messages also occur in normal news flow, so most hosts will already be suitably defended against undesired effects.
Published specification:	[USEFOR]
Body part:	A complete article or proto-article, ready for injection into Netnews, or a batch of such articles.

NOTE: It is likely that the recipient of an "application/news-transmission" will be a specialised gateway (e.g. a moderator's submission address) able to accept articles with only one of the three parameter types "moderate", "inject" and "relay", hence the reason why they are optional, being redundant in most situations. Nevertheless, they MAY be used to signify the originator's intention with regard to the transmission, so removing any possible doubt.

When the parameter "relay" is used, or implied, the body part MAY be a batch of articles to be transmitted together, in which case the following syntax MUST be used.

```
batch        = 1*( batch-header article )
batch-header = "#!" SPACE "rnews" SPACE article-size CRLF
article-size = 1*digit
```

where the "rnews" is case-sensitive. Thus a batch is a sequence of articles, each prefixed by a header line that includes its size. The article-size is a decimal count of the octets in the article, counting each CRLF as one octet regardless of how it is actually represented.

NOTE: Despite the similarity of this format to an executable UNIX script, it is EXTREMELY unwise to feed such a batch into a command interpreter in anticipation of it running a command named "rnews"; the security implications of so doing would be disastrous.

```
6.15.5.2 Application/news-message
[WARNING: The application/mews-message type as described here has been
subject to much adverse criticism. Thus it is liable to be replaced by
an alternative method of encapsulation in a future draft of this
document.]
        The Content-Type "application/news-message" is intended for
        the encapsulation of complete news articles which have already
        been posted to Netnews and which are for the information of
        the recipient, and do not constitute a request to repost them.
        This Message type SHOULD be used when a courtesy copy of a
        followup is mailed to the author of its precursor, and when
        gatewaying from news-to-mail.
```

Script approve.hsc

```
if (ParamStr(1)<>"mail.imap.approvemod")
  quit
endif

varset($Filename, ParamStr(2))
varset($Art, ArtAlloc)
if (ArtLoad($Art, $FileName) = 0)
  varset($MID, "<"+ copy(digest(2, Str(Random(100000))+GetProcessIdentifier,1),
1, 100) + "@" + HamMainFQDN + ">")
  ArtSetHeader($Art, "APPROVED:",
"<moderator@hamster2.beslernl.goldpool.org>")
  ArtSetHeader($Art, "SENDER:",
"<moderator@hamster2.beslernl.goldpool.org>")
  ArtSetHeader($Art, "X-OLD-DATE:",       ArtGetHeader($Art, "Date:"))
  ArtSetHeader($Art, "X-OLD-Message-ID:", ArtGetHeader($Art, "Message-ID:"))
  ArtSetHeader($Art, "Message-ID:",       $MID)
  ArtSetHeader($Art, "Date:",             ZeitFormat("mail", 0))

  ##Hier kann noch anderes getan werden, z.B. die Receive-Header löschen et al
  ArtSave($Art, $FileName)

  varset($list, listalloc(false, true))
  ListLoad($list, $FileName)
  ListBox($list, "")
  ListFree($List)
endif
ArtFree($Art)

quit


################################################################################
#Kopiert aus dem Script Meldung.hsc aus dem dauerlauf-Paket (Scriptarchiv)
#Bestätigung für die Verwendung steht noch aus...
Sub ZeitFormat($Typ, $UtVar)
```

Script moderation.hsc

```
if (ParamStr(1) != "n2m.moderate")
  quit
endif
varset($FileName, ParamStr(2))
varset($Result, ListAlloc(false, true))
varset($Art, ArtAlloc)

  if ((ListLoad($Result, $FileName)=0) && (ArtLoad($Art, $FileName)=0))
    varset($MID, "<"+ copy(digest(2, Str(Random(100000)) _
          + GetProcessIdentifier,1), 1, 100) + "@" + HamMainFQDN + ">")
    varset($boundary, "------_" _
         + copy(digest(2, Str(Random(100000))+GetProcessIdentifier,1), 1, 20) _
         + "_MULTIPART_MIXED_")

    #"Add" baut von unten nach oben auf, daher in anderer Reihenfolge und der
    #Haupt-"Umschlag" am Ende
    #Ganz am Ende muß erst einmal wegen multipart der boundary eingefügt werden
    ListAdd($Result, "--" + $boundary)
    #news-transmission-Umschlag um die eigentliche Nachricht
    Add("")
    Add("Content-Transfer-Encoding: " _
        + ArtGetHeader($Art, "Content-Transfer-Encoding"))
    Add("Content-Description: " + ArtGetHeader($Art, "Subject"))
    Add("Content-Type: application/news-transmission")
    Add("--" + $boundary)
    #Kommentar/Plain-Text Teil
    Add("")
    Add("sollte *nicht* publiziert worden sein. Sorry für die Umstaende")
    Add("Dies ist Bestandteil einer experimentellen IMAP/NNTP-Erweiterung und")
    Add("")
    Add("Dieser Kommentar wurde automatisch vom sendenden Hamster generiert.")
    Add("Diese Nachricht sollte moderiert werden.")
    Add("Content-Transfer-Encoding: 7bit")
    Add("Content-Type: text/plain; charset=""US-ASCII""")
    Add("--" + $boundary)
    #Envelope
    Add("")
    Add("MIME-Version: 1.0")
    Add("Content-Transfer-Encoding: 8bit")
    Add("X-Mailer: Hamster N2M-Moderation")
    Add("X-MODERATE: YES")
    Add("From: "       + ArtGetHeader($Art, "From:"))
    Add("To: "         + ArtGetHeader($Art, "To:"))
    Add("Subject: "    + ArtGetHeader($Art, "Subject:"))
    Add("Newsgroups: " + ArtGetHeader($Art, "Newsgroups:"))
    Add("Message-ID: " + $MID)
    Add("Date: "       + ZeitFormat("mail", 0))
    Add("Content-Type: multipart/mixed; boundary=""" + $boundary + """")
    #Speichere die geänderte Mail
    ListSave($Result, $FileName)
  endif
ArtFree($Art)
ListFree($Result)
quit

sub Add($s)
  ListInsert($Result, 0, $s)
endsub
Sub ZeitFormat($Typ, $UtVar)
#Kopiert aus dem Script Meldung.hsc aus dem dauerlauf-Paket (Scriptarchiv)
```

Erweiterung des Hamster-Quellcodes

```
################################################################################
---cIMAPMailbox.pas
type
  tOnApprove = procedure (Number : Integer) of Object; //Moder

  tIMAPNotification = record //HSR //IDLE
    OnNewMess  : tOnNewMess;
    OnExpunge  : tOnExpunge;
    OnApprove  : tOnApprove;//Moder
  end;

  TImapMailbox = class
    private
      function  Approve(const MsgFile : String;
                        const GroupPermission : tFuncPerm) : Boolean;
    public
      procedure ApproveMessages(const GroupPermission : tFuncPerm);

//------------------------------------------------------------------------------
implementation

uses cActions, cArtFiles; //Moder

const
     FLAGAPPROVE  : TFlagMask = 64;  //Moder


function TImapMailbox.StringToFlagMask( Flags: String ): TFlagMask;

     if pos( '\APPROVE',  Flags ) > 0 then Result := Result or FLAGAPPROVE;
end;

function TImapMailbox.FlagMaskToString( FlagMask: TFlagMask ): String;
begin
     Result := '';
     if FlagMask and FLAGAPPROVE = FLAGAPPROVE  then Result:=Result+' \Approve';


function TImapMailbox.GetPossFlags: String;
begin
  Result := '(\Answered \Flagged \Deleted \Seen \Draft \Approve)'
end;


procedure TImapMailbox.ApproveMessages(const GroupPermission : tFuncPerm);
var  Chunk : Byte;
     i,j   : Integer;
     MR    : TMessageRec;
     ServerThread : pIMAPNotification;
     FN: String;
begin
   if fReadOnly then exit;
   try
     Lock;
      for Chunk:=0 to fIndex.propCHUNK_MAX do begin
         fIndex.Enter( Chunk );
         try
            for i := fIndex.Count-1 downto 0 do begin
```

```
                fIndex.pubRecGet( Chunk, i, MR );
                if (MR.Flags and FLAGAPPROVE) = FLAGAPPROVE then begin
                  FN := fPath + IntToStr(SwitchByteOrder(MR.UID)) + '.msg';
                  if Approve(FN, GroupPermission) then
                    if fIndex.ContainsKey(MR, INDEXKEYLEN_UID ) then begin
                      if (MR.Flags and FLAGSEEN)<>FLAGSEEN then
                        dec(fStatus.Unseen);
                      if (MR.Flags and FLAGRECENT)=FLAGRECENT then
                        dec(fStatus.Recent);

                      fIndex.RemoveFlags(i, FLAGAPPROVE);// OR FLAGRECENT);
                      fIndex.AddFlags(i, FLAGDELETED OR FLAGSEEN);
                      //ToDo: Vielleicht doch ein neuer Flag?

                      for j := 0 to fUsers.Count-1 do try
                        if Assigned( fUsers.Items[j] ) then begin
                          ServerThread := pIMAPNotification( fUsers.Items[j] );
                          if Assigned(ServerThread) then
                            if Assigned(ServerThread^.OnApprove) then
                              ServerThread^.OnApprove( i+1 )
                        end
                      except end
                    end
                end
              end
            finally
              fIndex.Leave(Chunk)
            end
        end;
        fIndex.SaveToFile
    finally
        Unlock
    end;
    WriteStatus
end;
```

```
function TImapMailbox.Approve(const MsgFile: String;
                              const GroupPermission : tFuncPerm): Boolean;
var Art : tIMAPMessage;
    NRRes : tNewsRouterCHKResult;
    ClInf : tClientInfos;
    s, DataBuf : string;
begin
  Result := false;
  Art := tIMAPMessage.Create;
  try
    Log(LOGID_DEBUG, MsgFile);
    if FileExists2(MsgFile) then begin
      Art.LoadFromFile(MsgFile);
      Art.Text := Art.GetModerationPart; //Get only the Moderation-Part
                                         //(last if more than one)

      DataBuf := Art.Text;
      // Action for groups go local
      If Actions.Exists ( atIMAPApproveMod ) then begin
        Case ModifyMessage( atIMAPApproveMod, DataBuf, s) of
          mcOriginal: ;
          mcChanged: Art.Text := s;
          mcDeleted: exit
        end
      end;

//      MessageBox(0, PChar(s), 's', 0);

      NRRes := NewsrouterCheck(self, GroupPermission, Art, titIMAPModerate);
      If NRRes.Err=[] then begin
        if NRRes.CheckedHdrs.NGsGateway  <>'' then exit;
           //Keine Gateway-Gruppen erlaubt
        if NRRes.CheckedHdrs.NGsModerated<>'' then exit;
           //Kann nicht, da Approved-Header gesetzt

        ClInf.RemAddrInt := 0; //ToDo: ClientInfo (IMAP)
        ClInf.RemAddr := '';
        ClInf.UserID := -1;
        Log(LOGID_DEBUG, 'BEGIN APPROVING');

        //## R E M O T E ##
        Result := NewsRouterSendRemote(NRRes.CheckedHdrs.NGsRemote,
                                       Art, ClInf, NRRes.CheckedHdrs).Err=[];

        // if local injection is not used then exit
        if (not Result) OR ((not Def_News_LocalInjection)
           and (NRRes.CheckedHdrs.NGsLocal='')) then exit;

        Log(LOGID_DEBUG, 'LOCAL APPROVING');
        //## L O C A L ##
        if NRRes.CheckedHdrs.NGsLocal<>'' then
          Result := NewsRouterSendLocal(NRRes.CheckedHdrs.NGsLocal,
                      Art, true, false, true, CLInf, NRRes.CheckedHdrs).Err=[];
      end; //Result = false!
    end;
  finally
    Art.free;
  end;
end;
```

```
################################################################################
---cServerIMAP.pas
type
  TSrvIMAPCli = class(TSrvWorkCli)
    private
      SendApprove     : array of Integer; //Moder

      function ModGrpPerm(Groupname : String) : Boolean;
      procedure NewApprove(Number : Integer); //Moder

IMPLEMENTATION

procedure TSrvIMAPCli.SendResTag( Txt: String);
var i : integer;
begin
  {HSR} {IDLE}
  try
    EnterCriticalSection( CS_THR_IDLE );
    if Assigned(Selected) then try
      Selected.Lock;
      if (length(SendExpunge)>0) then begin
        for i := 0 to length(SendExpunge)-1 do
          SendRes( IntToStr(SendExpunge[i]) + ' EXPUNGE');
        for i := 0 to length(SendApprove)-1 do //Moder
          SendRes( IntToStr(SendApprove[i]) + ' APPROVE');

        if not SendNewMessages then //Don't send it double
          SendRes( IntToStr(Selected.Status.Messages) + ' EXISTS');
        SetLength(SendExpunge, 0);
        SetLength(SendApprove, 0) //Moder
      end;


procedure TSrvIMAPCli.HandleCommand( Const CmdLine: String );

        if Cmd='UID'      then begin Cmd_UID     ( Par ); exit end;
        if Cmd='APPROVE'  then begin Cmd_Approve ( Par ); exit end; //Moder

        // unknown (sub-) command

procedure TSrvIMAPCli.Cmd_CAPA( Par: String );

  //---Standard-CAPAs-------------------------------
  capabilities := 'IMAP4rev1 '
                + 'AUTH=CRAM-SHA1 '
                + 'AUTH=CRAM-MD5 '   //JW //IMAP-Auth
                + 'AUTH=DIGEST-MD5 ' //JW //SASL-DIGEST
                + 'IDLE '            //HSR //IDLE
                + 'LITERAL+ '        //HSR //Literal+
                + 'XMODERATE '; //Moder
```

```
procedure TSrvIMAPCli.DoList( Par: String; LSub: Boolean );

     procedure SendList( Txt: String );
     var  s: String;
     begin
          if LSub then s := 'LSUB (' else s := 'LIST (';
          if FileExists2( MailBoxPath + ReplacePathDelimiters( Txt )
                          + 'NOSELECT' )
             then s := s + '\NOSELECT ';
          if FileExists2( MailBoxPath + ReplacePathDelimiters( Txt )
                          + 'MODERATE' )
             then s := s + '\XMODERATED '; //Moder

          SendRes( trim(s) + ') "' + HierarchyDelimiter + '" "' + Txt + '"' );
     end;


procedure TSrvIMAPCli.Cmd_APPROVE(Par: String); //Moder
begin
  if Par <> '' then begin
    Log(LOGID_WARN, 'IMAP: APPROVE: Too many arguments!');
    SendResTag('BAD I don''t know parameters for APPROVE!')
  end else begin
    if Assigned( Selected ) then begin
      if FileExists2(IncludeTrailingBackslash(selected.Path)+'MODERATE') then
      begin
        Selected.ApproveMessages(self.ModGrpPerm);
        SendResTag ('OK APPROVE completed')
      end else
        SendResTag ('NO APPROVE: Folder not Moderated')
    end else
      SendResTag ('NO APPROVE: No Folder selected');
    Log(LOGID_DETAIL, 'IMAP: APPROVE');
  end;
end;

procedure TSrvIMAPCli.NewApprove(Number: Integer); //Moder
begin
  try
    EnterCriticalSection( CS_THR_IDLE );
    if IdleState then begin
      if Assigned(Selected) then begin
        SendRes( IntToStr(Number) + ' APPROVE')
       end
    end else begin
      SetLength(SendApprove, length(SendApprove)+1);
      SendApprove[length(SendApprove)-1] := Number
    end
  finally
    LeaveCriticalSection( CS_THR_IDLE )
  end
end;

function TSrvIMAPCli.ModGrpPerm(Groupname: String): Boolean; //Moder
begin //Temporär für Bes. Lernl.
  Result := true
end;
```

```
################################################################################
---cIMAPMessage.pas
type
  TImapMessage = class( TArticle )
    public
      function GetModerationPart : String; //Moder



function TImapMessage.GetModerationPart: String;  //Moder
var i : integer; CRLFFound, j : Integer;
begin
  Result := '';
  if (ContentType = 'MULTIPART') then
    for i := 0 to High( Parts ) do begin
      if (Parts[i].ContentType = 'APPLICATION') and
         (Parts[i].ContentSubType = 'NEWS-TRANSMISSION') then
      begin
        Result := Parts[i].Text
      end
    end
  else if (ContentType = 'APPLICATION') and
          (ContentSubType = 'NEWS-TRANSMISSION') then
       begin
          Result := Text
       end;

  //Strip the first four lines
  CRLFFound := 0;
  j := 1;
  while ((j<length(Result)) and (CRLFFound<4)) do begin
    if (Result[j]=#13) and (Result[j+1]=#10) then
      inc(CRLFFound);
    inc(j)
  end;
  if CRLFFound=4 then
    Result := copy(Result, j+1, length(Result))
  else
    Result := ''
end;
```

```
################################################################################
//---cNewsrouter.pas: Verschiedene Änderungen, nicht mehr genau herauslesbar
//(Infolge des Umbaus für den Newsrouter gab es auch Änderungen in
cServerNNTP.pas)

################################################################################
---cActions.pas
Type
  TActiontype =
       atMailHeader, atMail, atMailIn, atMailInternal, atMailLocal, atMailOut,
       atIMAP, atIMAPApproveMod, //Moder
       atNTM, atNTMGateway, atNTMModerate, //HSR //Newsrouter

Const
  ActionIniKeys: Array[TActiontype] of String
       'mail.getheader', 'mail', 'mail.in', 'mail.internal', 'mail.local',
       'mail.imap', 'mail.imap.approvemod', //Moder
       'n2m', 'n2m.gateway', 'n2m.moderate', //HSR //Newsrouter

  ActionParent: Array[TActiontype] of TActiontype =
       atNone, atNone, atMail, atMail, atMail, atMail,
       atNone, atIMAP, //Moder
       atNone, atNTM, atNTM, //HSR //Newsrouter

################################################################################
---HConfigAutomatic.pas
      Add ( 2, atMailOut, false, 'OutgoingMail', 'Outgoing Mail',
         'This action is executed for every Mail to an external mailadress. '
          + 'Parameter contains the filename.' );

      //Moder
      Add ( 0, atIMAP, false, 'Mail-IMAP', 'IMAP-Mail',
         'This action is executed on every mail to an IMAP-account (not yet!!) '
          + 'Parameter contains the filename.' );
      Add ( 1, atIMAPApproveMod, false, 'Mail-IMAP: ApproveMod',
         'IMAP: Approve moderated', ,This action is executed for every '
        +'posting approved with IMAP '
        +'Parameter contains the filename.' );

      //HSR //Newsrouter
      Add ( 0, atNTM, false, 'News-To-Mail', 'News-To-Mail',
         'This action is executed on every posting to a moderated or '
        +'gateway newsgroup '
        +'Parameter contains the filename.' );
      Add ( 1, atNTMGateway, false, 'Gateway', 'News-To-Mail Gateway',
         'This action is executed for every posting to a gateway-newsgroup '
          + 'Parameter contains the filename.' );
      Add ( 1, atNTMModerate, false, 'Moderation', 'News-To-Mail Moderation',
         'This action is executed for every posting to a moderated '
        +'newsgroup without approve '
        +'Parameter contains the filename.' );
      //HSR
```