UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

Projeto de um Protocolo para a Recuperação Eficiente de Informação em Caixas Postais Eletrônicas

Fellipe Araújo Aleixo

Campina Grande – PB
1999

Universidade Federal da Paraíba – UFPB
Centro de Ciências e Tecnologia – CCT
Coordenação de Pós-graduação em Informática - COPIN

Projeto de um Protocolo para a Recuperação Eficiente de Informação em Caixas Postais Eletrônicas

Fellipe Araújo Aleixo

Campina Grande
1999

Universidade Federal da Paraíba – UFPB
Centro de Ciências e Tecnologia – CCT
Coordenação de Pós-graduação em Informática - COPIN

FELLIPE ARAÚJO ALEIXO

Projeto de um Protocolo para a Recuperação Eficiente de Informação em Caixas Postais Eletrônicas

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-graduação em Informática da Universidade Federal da Paraíba – Campus II, como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

Orientador: **Prof. Francisco Vilar Brasileiro, Ph.D**
Linhas de Pesquisas: **Sistemas Distribuídos e Redes de Computadores**
Área de Concentração: **Ciência da Computação**

Campina Grande
1999

ALEIXO, Fellipe Araújo

A366P

Projeto de um Protocolo para a Recuperação Eficiente de Informação em Caixas Postais Eletrônicas

Dissertação (Mestrado) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, Coordenação de Pós-graduação em Informática, Campina Grande, Outubro de 1999.

122p. Il.

Orientador: Prof. Francisco Vilar Brasileiro, Ph.D

Palavras chave:

1. Redes de Computadores
2. Sistemas Distribuídos
3. Correio Eletrônico
4. Acesse e Recuperação de Mensagens de Correio Eletrônico

CDU - 621.391

ESPECIFICAÇÃO DE UM PROTOCOLO PARA RECUPERAÇÃO EFICIENTE DE INFORMAÇÃO EM CAIXAS POSTAIS ELETRÔNICAS

FELLIPE ARAÚJO ALEIXO

DISSERTAÇÃO APROVADA EM 24.08.1999

PROF. FRANCISCO VILAR BRASILEIRO, Dr.
Orientador

PROF. CARLOS ANDRÉ GUIMARÃES FERRAZ, Ph.D
Examinador

PROF. JACQUES PHILIPPE SAUVÉ, Ph.D
Examinador

CAMPINA GRANDE – PB

A Deus, toda a honra e glória.
Aos meus pais e minha irmã.
A Isabel.
Ao meu amigo e orientador, Fubica.
Aos meus amigos, essa vitória também é de vocês.

Resumo

Este trabalho apresenta a proposta de um novo protocolo para o acesso e recuperação de mensagens de Correio Eletrônico Internet. Correntemente, os protocolos mais conhecidos que realizam esta tarefa são o POP e o IMAP. Nenhum destes dois protocolos oferece alguma funcionalidade que auxilie a administração automática das Caixas Postais Eletrônicas. Este novo protocolo, denominado de *Smart Post Office Protocol – SPOP*, armazena as mensagens no servidor, vinculadas a pastas especiais de arquivamento, denominadas *Smart Folders* ou *SFolders*. Cada *SFolder* tem associado a si uma regra que estipula as características que devem possuir as mensagens para estarem vinculadas àquela pasta. O trabalho também define uma linguagem utilizada para a definição das regras de formação dos *SFolders* denominada *Smart Folder Definition Language (SFDL)*. Uma vez criados os *SFolders*, o processo de classificação das mensagens acontece de forma automática, o que garante a consistência da classificação das mensagens cada vez que o cliente se conectar à referida caixa postal eletrônica. Encontrando as mensagens da caixa postal eletrônica classificadas por critérios específicos, a recuperação acontece de forma rápida e objetiva.

Abstract

This work presents a proposal for a new protocol for the access and retrieval of messages in an Internet eletronic mail system. Currently there are two protocols that can be used to access and retrieve messages. This kind of work is made by the POP and IMAP protocols. None of these protocols offer functionality that helps in the automatic administration of electronic mailboxes. A new protocol, named *Smart Post Office Protocol – SPOP*, stores the messages in the mail server, associated with special messages folders, called *Smart Folders* or *SFolders*. Each *SFolder* has a rule that specifies the kind of messages that it will store. This work also defines a language that is used to contruct the rules of a *SFolder*, called *Smart Folder Definition Language (SFDL)*. Once the *SFolder* has been created, the entire message classification process is performed automatically, ensuring the consistency of the classification whenever the client accesses the mailbox. The retrieval of messages can then be performed efficiently.

Agradecimentos

Ao meu Deus, que me concedeu a graça de conhecê-lo e de mudar a minha vida, e que me ajudou a concluir a primeira fase deste projeto. E que sempre se apresentou como Pai, nos momentos que eu mais precisei de força e direção, a Ele todo o louvor, honra e glória.

Aos meu pais, Adalberto e Irismar, por todas as condições oferecidas para que conseguisse chegar hoje onde estou e ser quem sou.

À minha irmã e amiga, Lidianne, por saber que sempre posso contar com você.

Às minhas maiores e melhores amigas, Renata e Katiane, junto de quem pude experimentar como uma corda de três dobras não se quebra facilmente.

Ao meu orientador e amigo, Fubica, pelo zêlo e cuidado com o meu trabalho, além do encorajamento (nada convencional).

À Isabel, a bela que suportou a fera.

Ao LSD *Team*: Gustavo, Érica, Lívia, Tatiana, Luiza, Milton, José Augusto (Guga), e a toda *new generation* do LSD.

A todos que fazem o DSC e a COPIN.

A todos que oraram por mim, para que concluísse sem problemas este trabalho, que Deus os recompense sobremaneira.

Sumário

Índice das Figuras

Índice das Tabelas

Capítulo 1

Introdução

O sistema de correio eletrônico Internet[1] [Hughe98] vem se tornando o canal de comunicação mais usado em ambientes de trabalho ou residências. Muitos especialistas concordam que o correio eletrônico é a aplicação da Internet mais importante e difundida, embora a *World Wide Web* continue sendo vista como a mais promovida.

Algumas estimativas indicam que cerca de 30% de toda a correspondência que anteriormente seguia pelos meios tradicionais de correios, segue agora por correio eletrônico. Nos Estados Unidos, por exemplo, segundo dados do Departamento de Comércio, no ano de 1997, foram enviadas mais mensagens eletrônicas do que cartas através do sistema de correio tradicional[2].

O desenvolvimento mais acentuado da área de correio eletrônico se deu nos últimos anos, como parte da explosão da Internet. Antes deste real desenvolvimento da tecnologia de correio eletrônico, tivemos sistemas de correio eletrônico que perduraram por mais de dez anos sem sofrer nenhuma alteração nas suas formas de funcionamento. Para dar suporte a este crescimento, muitos pesquisadores têm investigado as deficiências dos atuais padrões que especificam a arquitetura do sistema de correio eletrônico, propondo novas soluções, com novas e melhores formas de abordagem dos problemas.

1.1. Breve Histórico do Serviço de Correio Eletrônico

O serviço de correio eletrônico nasceu com a tecnologia de redes de computadores [Tanen88]. Primeiro se conseguiu interligar os computadores, e possibilitar a comunicação entre os mesmos. O segundo passo foi possibilitar aos usuários um serviço para enviar e receber correspondências eletrônicas através

[1] Nome utilizado para denominar a rede mundial de computadores, que tem os protocolos de comunicação TCP e IP (*Transfer Control Protocol* e *Internet Protocol*) [Steve94] [Wrigh95] como base da sua estrutura.

[2] Fonte: USA Today, 17 de Março de 1999.

destes computadores, conectados à rede. Foi criada, desta forma, uma importante tecnologia de comunicação interpessoal.

As primeiras implementações do serviço de correio eletrônico surgiram há mais de duas décadas. No início, este serviço era baseado na transferência de arquivos de uma máquina para outra. Com o tempo, ficaram evidentes as limitações deste enfoque, como por exemplo: i) era muito inconveniente enviar uma mesma mensagem para um grupo de pessoas; ii) mensagens não tinham uma estrutura interna; iii) o remetente nunca podia ter certeza se a mensagem havia chegado sem danos ao destinatário; iv) caso um usuário estivesse planejando se ausentar por várias semanas e desejasse que as mensagens, recebidas nesse período, fossem encaminhadas para a sua secretária, não seria fácil conseguir isto; v) a interface com o usuário estava pouco integrada ao sistema de transmissão, já que era preciso antes, editar o arquivo em um editor de texto comum, e só então iniciar o programa encarregado de realizar a transferência do arquivo. Além dessas limitações, apenas texto ASCII podia ser transmitido.

Com o tempo, novas propostas surgiram e o serviço de correio eletrônico cresceu em importância. Estes fatos levaram instituições especializadas em padronização a buscarem a especificação de uma estrutura e uma funcionalidade padrão para o serviço de correio eletrônico. Em 1984, o CCITT[3] (*Consultative Committee for International Telegraphy and Telephony)* elaborou sua série de recomendações denominadas de X.400, que é o padrão de correio eletrônico que especifica o MHS (*Message Handling System*), o sistema de manipulação de mensagens. O X.400 é o padrão de correio eletrônico que funciona sobre o padrão de transporte em rede X.25. O X.400 mais tarde foi usado como base para o MOTIS (*Message-Oriented Text Interchange System*) da ISO (*International Organization for Standardization*). Em 1988, o CCITT modificou o X.400 para ajustar-se ao MOTIS.

Com o surgimento da arquitetura TCP/IP (*Transfer Control Protocol / Internet Protocol*) [Steve94] [Wrigh95], possibilitou-se a criação da rede mundial de computadores: a Internet. Utilizando a infra-estrutura da Internet surgiu também um conjunto de padrões para especificar o serviço de correio eletrônico na Internet. Os principais padrões utilizados pela arquitetura do correio eletrônico Internet são o SMTP (*Simple Mail Transfer Protocol*) [Poste82], o POP (*Post Office Protocol*) [Myers96], o IMAP (*Intenet Message Access Protocol*) [Crisp96] e o MIME (*Multipurpose Internet Mail Extension*) [Freed96] [Klens96] [Moore96] [Boren96] [Freed96-2] [Nelso97]. O

[3] Antigo nome do ITU-T (International Telecommunications Union, Telecommunication Standards Sector), entidade que coordena os serviços e padrões internacionais de telecomunicação.

protocolo SMTP define como é feito transporte das mensagens da origem ao destino. O POP e o IMAP definem como acontece o acesso dos usuários a suas mensagens. O MIME estabelece um conjunto de extensões ao formato original das mensagens definido pelo SMTP. Uma ilustração de como se dá a interação entre os protocolos que fazem parte do Sistema de Correio Eletrônico Internet, pode ser encontrado no Apêndice A deste documento.

Até recentemente, a maioria dos sistemas comerciais de correio eletrônico eram desenvolvidos com tecnologia proprietária. Parte destes sistemas de correio proprietários, são distribuídos com os ambientes operacionais, como os da Microsoft, o Microsoft Mail [Catap94] [Flynn93] e o Microsoft Exchange [Nelso96] [Kilcu96]. Outra parte dos sistemas de correio eletrônico com tecnologia proprietária fazem parte de pacotes de aplicativos, com o objetivo de auxiliar gerenciar as informações para grupos de trabalho; nesta categoria podemos citar o exemplo do Lotus Notes [Notes93]. Uma das grandes ferramentas que o Notes possui é a base de dados[4] "Mail", que é o seu sistema de Correio Eletrônico. Com o amadurecimento dos padrões na área de correio eletrônico, a tendência que se observa é a migração dos sistemas de correio eletrônicos proprietários para a tecnologia de correio eletrônico nativa da Internet.

1.2. Estrutura do Serviço de Correio Eletrônico

O MOTIS foi o primeiro padrão genérico para a arquitetura de correio eletrônico a ser aceito tanto pelos usuários quanto pelos desenvolvedores de aplicações para este serviço. O MOTIS foi definido pela ISO através do documento ISO 10021 [ISO90] [ISO90-2] [ISO90-3]. Esse documento determina o comportamento da arquitetura e formaliza um conjunto de entidades, bem como o relacionamento entre estas entidades. Explicaremos qual o papel de cada entidade no modelo MOTIS e em seguida mostraremos uma figura que explicitará a forma de interação entre as mesmas. Como a nomenclatura definida pelo MOTIS tornou-se um padrão para o ambiente de correio eletrônico, as entidades definidas pelo MOTIS podem ser mapeadas nas entidades de outros sistemas cliente/servidor de correio eletrônico. No decorrer deste documento, quando tratarmos do modelo de correio eletrônico da Internet, faremos referências às entidades aqui definidas.

[4] O Lotus Notes é um gerenciador de Informações para grupos de trabalho que, com dois objetivos principais: compartilhar informações e facilitar a comunicação. Todo o seu trabalho gira em torno do gerenciamento de grandes bases de dados que são compartilhadas por vários usuários.

A primeira entidade de que trataremos é o Agente de Correio do Usuário (*Mail User Agent*), ou simplesmente UA. O UA, que também é chamado de cliente de correio eletrônico, é o programa de computador através do qual o usuário tem acesso ao sistema de correio eletrônico. O UA é o ponto primário de entrada e saída do sistema de correio eletrônico, onde o usuário pode ler as suas mensagens, manipula-las como desejar, bem como compor e enviar mensagens para outros usuários. O UA equivale a um aparelho de telefone em um sistema de telefonia. Um UA pode ter a forma de qualquer aplicação com a habilidade de acessar o sistema de correio eletrônico. O UA só é executado quando o usuário deseja ler ou compor mensagens, ou checar a existência de novas mensagens.

A segunda entidade é o Agente de Transferência de Mensagens (*Message Transfer Agent*), ou simplesmente MTA. Um conjunto de MTAs constitui o próprio sistema de correio eletrônico. O conjunto de MTAs é responsável por coletar as mensagens eletrônicas dos UAs de origem, rotear cada mensagem através de outros MTAs até chegar a um MTA de destino e por fim, distribuir as mensagens para os UAs destino, quando estes as solicitarem. O MTA encaminha as mensagens para outros MTAs, no caso das mensagens trocadas entre usuários em diferentes servidores de correio. O MTA equivale a uma central de comutação telefônica em um sistema de telefonia. Os MTAs têm de estar funcionando sempre, prontos para receber conexões de UAs e outros MTAs. O conjunto de todos os MTAs espalhados pela rede forma o próprio sistema de transferência de mensagens, que é encarregado de transferir uma mensagem da origem ao destino.

A terceira entidade chama-se Repositório de Mensagens (*Message Store*), ou simplesmente MS. O MS também é conhecido como a central de caixas de correio. Esta entidade tem a forma de um banco de dados textual, sem formato especificado, que é utilizado pelos MTAs para armazenar as mensagens recebidas dos UAs ou outros MTAs, destinadas a um usuário cadastrado no MTA em questão. O MS é a entidade que centraliza o acesso e recuperação de mensagens. Não é responsabilidade do MS guardar qualquer histórico sobre o acesso e recuperação das mensagens. Caso necessário, é o UA que deve manter o registro das mensagens que foram recuperadas pelo mesmo. A Figura 1 [Tanem88] abaixo mostra a interação dos elementos nesse modelo.

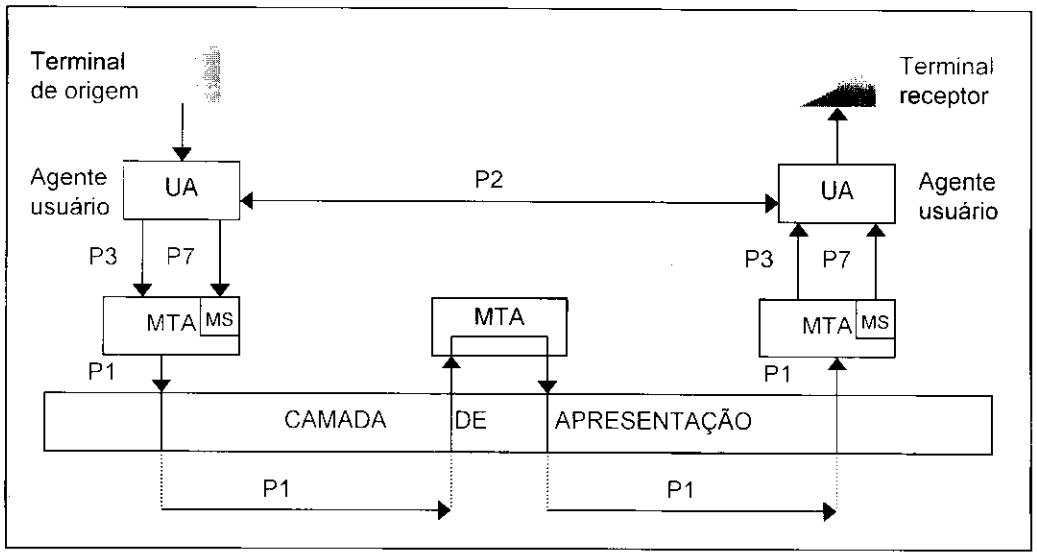

Figura 1.1 – MOTIS (Message Oriented Text Interchange System).

O MS estabelece uma interface padrão para armazenamento e recuperação de mensagens. Esta interface normalmente é implementada pelo MTA, mas pode ser implementada por um servidor específico. O MS, mesmo tendo um controle centralizado, pode utilizar várias máquinas para guardar as mensagens, objetivando prover soluções escaláveis que atendam a um grande número de usuários. Dá-se o nome de Agência de Correio Eletrônico a quem responde pelos serviços estipulados pelas entidades MTA e MS.

Os vários rótulos Pn representam os protocolos de comunicação utilizados entre as entidades. O rótulo P2 representa o protocolo virtual que define o cabeçalho e o corpo de uma mensagem eletrônica, virtual porque trata-se apenas de uma convenção. Já o rótulo P1 representa o protocolo que define e manipula os envelopes, que são as estruturas que encapsulam o cabeçalho e o corpo de mensagens. Os envelopes contêm as informações necessárias para que a mensagem seja roteada com sucesso até o MTA de destino. O rótulo P3 representa o protocolo de comunicação entre o UA e o MTA, que é utilizado tanto para enviar mensagens quanto para recuperar as informações da caixa postal do usuário. O rótulo P7 simboliza o protocolo de comunicação entre o UA e o MS, que é utilizado tanto para a recuperação de mensagens quanto para a transferência de uma mensagem para o MS, enquanto ela não é enviada para o seu destino pelo MTA.

1.2.1. Estrutura do Serviço de Correio Eletrônico Internet

Um conjunto de protocolos primários define o funcionamento do serviço de correio eletrônico na Internet. Estes protocolos primários são: SMTP, POP, IMAP e MIME. O SMTP é o protocolo fundamental para o serviço de correio eletrônico

Internet. O SMTP foi formalizado através das RFC[5]s 821 [Poste82] e 822 [Crock82], que definem o estilo do correio eletrônico Internet. O SMTP é o protocolo utilizado tanto pelo MTA para trocar mensagens com outros MTAs, como também pelo UA para enviar mensagens. Existem também outras RFCs que complementam o SMTP [Winte96] [Moore96-2] [Klens95] [Freed95] [Cargi85] [Crock95] [Vaudr95] [Klens94] [Freed94] [Rose94]. Estas RFCs definem extensões à definição inicial do SMTP e visam adaptar e adequar o SMTP aos avanços da tecnologia, buscando resolver as deficiências no modelo inicial.

Os protocolos POP e IMAP são utilizados pelo UA para o acesso e recuperação de mensagens de correio eletrônico armazenadas em um MS (eqüivalem ao P7, na descrição que fizemos do MOTIS). O acesso se dá tanto às informações da caixa de correio do usuário, quanto às informações individuais de cada mensagem.

O POP é o protocolo mais utilizado pelos UAs para recuperar mensagens armazenadas em um MS. As RFCs 1082 [Rose88-2] e 1460 [Rose93] definem este protocolo. O POP especifica como deve ser a interação entre um dado UA e um servidor que acessa o MS de uma Agência de Correio Eletrônico Internet, ambos devendo suportar o protocolo POP. O POP foi desenvolvido para possibilitar a um UA acessar as mensagens em um MS e efetuar a transferência das mensagens para o disco local na máquina onde o UA está executando e posteriormente apagar as mensagens do MS.

O IMAP é um protocolo que faz parte de uma geração mais recente de protocolos. Foi desenvolvido para suprir algumas carências do protocolo utilizado até então para prover o acesso e recuperação às mensagens de correio eletrônico, o POP. As carências que o IMAP veio atender foram: a pesquisa por mensagens com determinadas características, bem como a centralização do armazenamento das mensagens, para possibilitar o acesso de forma distribuída. O IMAP é utilizado para gerenciar o acesso às mensagens de correio eletrônico armazenadas em um servidor MS de uma agência de correio eletrônico. O IMAP foi originalmente definido pelas RFCs 1730 [Crisp94-3], 1731 [Myers94-2], 1732 [Crisp94-2] e 1733 [Crisp94].

O IMAP foi desenvolvido originalmente para acessar e manter as mensagens no MS central. O IMAP também possui funcionalidades adicionais, para possibilitar a pesquisa por mensagens que satisfaçam um determinado critério, recuperação apenas do cabeçalho das mensagens e recuperação seletiva de mensagens, que serão descritas com mais detalhes mais adiante neste documento.

[5] RFC é a sigla para *Request for Comments*. As RFCs são documentos, gerados pelo *Internet*

O MIME representa um conjunto de extensões à sintaxe original das mensagens definida pelo SMTP na RFC 822. Com o tempo, mensagens que continham apenas textos ASCII não eram mais suficientes para atender às exigências dos usuários, necessitava-se de algum meio para possibilitar mensagens mais ricas em conteúdo; o MIME surgiu para suprir essa deficiência. O MIME permite anexar vários tipos de arquivos (ASCII ou binários) ao texto da mensagem, além de suportar várias organizações para as mensagens (*multipart*, *parallel*, etc.). O MIME também define o suporte a um formato de texto com diferentes fontes, tamanho diferente de fontes e até cores diferentes. Os servidores podem ignorar as extensões completamente, visto que uma mensagem com um anexo MIME continua sendo uma mensagem no formato texto que segue a sintaxe original definida pela RFC 822, mesmo se um ou mais arquivos binários tiverem sido anexados.

1.3. Acesso e Recuperação de Mensagens

Abordaremos agora alguns aspectos relacionados com o acesso e recuperação de mensagens, que é a questão específica a ser tratada neste trabalho. Num primeiro momento, gostaríamos de resgatar uma afirmação que fizemos no início desta seção, onde apontávamos a existência de uma tendência dos usuários de sistemas de correio eletrônico, de uma forma geral, migrarem de suas soluções proprietárias para a tecnologia de correio eletrônico nativa da Internet.

Essa previsão de migração e a introdução dessa nova gama de usuários no sistema de correio Internet, ocasionará em primeira instância dois efeitos. O primeiro é o aumento considerável no volume das mensagens eletrônicas que irão trafegar pela rede (mais clientes enviando mensagens, ocasionando mais mensagens na rede). O segundo efeito também ligado ao aumento do volume de mensagens, é a valorização do meio e o aumento de importância das informações que circulam na forma de mensagens de correio eletrônico (o correio eletrônico se torna de missão crítica). Não só uma grande quantidade de informações importantes será transmitida na forma de mensagens de correio eletrônico, mas também boa parte dessas informações importantes serão armazenadas neste formato.

Com o aumento na quantidade, importância e valor da informação armazenada na forma de mensagens de correio eletrônico, fazem-se necessárias ferramentas para a recuperação mais eficiente destas informações. Para possibilitar a construção de tais ferramentas, torna-se evidente a necessidade de protocolos mais eficientes e flexíveis para suportar esta nova missão do acesso e recuperação de mensagens. A

Engineering Task Force (IETF), que definem os protocolos e políticas para a Internet.

"eficiência" que buscamos é o aumento da garantia que um usuário recuperará exatamente a informação que deseja, em um tempo mais curto possível. Se enquadraram nessa definição, protocolos com algum sistema de organização das mensagens, voltados a permitir que um usuário encontre rapidamente a informação que necessita no momento, sem que lhe seja adicionado nenhum ônus por tal facilidade.

Como poderá ser constatado na discussão a seguir, os protocolos de acesso e recuperação de mensagens do correio eletrônico Internet atualmente utilizados – POP e IMAP – não atendem a tais necessidades dos seus usuários.

O POP é o protocolo mais simples para tal fim, e como solução mais simples não adiciona nenhuma informação ao armazenamento das mensagens. O trabalho do POP é voltado à transferência das mensagens do MS para o UA. Porém, programas clientes de correio eletrônico que utilizam o POP podem realizar localmente a classificação das mensagens transferidas do MS. Esta "classificação" ocorre quando o usuário decide separar as suas mensagens de acordo com determinados critérios. Esta separação pode se dar de diferentes formas. Um exemplo de como é realizada a separação das mensagens é a tecnologia de filtros de mensagens (muito utilizada em programas clientes de correio eletrônico, como por exemplo o MS Outlook Express). Estes filtros de mensagens, após configurados, são responsáveis por inspecionar uma mensagem que acabou de ser transferida, armazenando-a em uma pasta específica, de acordo com alguns critérios predefinidos.

Existem dois problemas com essa abordagem dada pelos clientes POP. Em primeiro lugar, a classificação não afeta as mensagens previamente transferidas, significando que no caso de existirem mensagens previamente recuperadas a classificação será parcial. E em segundo lugar, se o usuário utilizar várias máquinas para manipular as suas mensagens ele terá que manter a consistência manualmente em cada cliente POP que ele utilizar, pois a definição de uma pasta com filtro de mensagens é local a cada UA.

Quando o usuário realmente necessita acessar as suas mensagens de correio eletrônico a partir de vários pontos da rede, encontra dificuldades para resolver o seu problema utilizando o POP. O POP não possui funcionalidades voltadas a possibilitar o acesso distribuído a mensagens. O IMAP foi desenvolvido originalmente para atender a essa necessidade específica, que é o acesso remoto e distribuído a mensagens de correio eletrônico. Na arquitetura do IMAP as mensagens de correio eletrônico

permanecem armazenadas em um servidor central e são acessadas remotamente pelos clientes IMAP.

O IMAP já oferece o conceito de pastas, que podem ser usadas como um mecanismo de classificação de mensagens. O IMAP possibilita ao usuário criar e manipular pastas de arquivamento de mensagens no servidor. Porém, todo o trabalho com as pastas é manual; o usuário tem que informar quais mensagens deseja mover para quais pastas. Outra funcionalidade voltada a atender a necessidade de recuperação eficiente de informações é oferecida pelo comando SEARCH do IMAP. Esse comando é utilizado para fazer pesquisa na caixa postal do usuário (no servidor) por mensagens que possuam um determinado critério, como por exemplo, mensagens que foram recebidas há "5" dias, foram enviadas pelo usuário "iris@ufrnet.ufrn.br" e possuem tamanho maior que "50" bytes.

Mesmo com o comando SEARCH o IMAP continua a deixar nas mãos do usuário a classificação de mensagens, mesmo que facilitando em muito o processo. Para utilizar o comando SEARCH para classificar mensagens em pastas, primeiro o usuário têm que formular o critério de busca para utilizar com o comando SEARCH. Segundo, deve criar uma pasta, e só então copiar todas as mensagens do resultado para a pasta criada. O problema com este tipo de abordagem é que não é garantida a consistência com as mensagens que ainda chegarão à caixa de correio do usuário, e para tal o usuário tem que realizar novamente o processo de busca na próxima vez que ele acessar a sua caixa postal.

Identificamos portanto, a necessidade de um serviço que possibilite a classificação das mensagens que um usuário possui em sua caixa de correio, localizada em um MS. Lavando em consideração que uma das maiores virtudes do POP é ser um protocolo extremamente simples, e que por este motivo ainda é um protocolo muito utilizado para o acesso e recuperação de mensagens. Da mesma forma, levando em consideração que o IMAP é um protocolo com muitas funcionalidades, e que por outro lado é muito complexo. A alternativa escolhida foi desenvolver um novo protocolo. Uma extensão do POP e uma alternativa ao IMAP. Objetivando que seja simples de implementar e de utilizar, que ofereça recursos de classificação programada de mensagens, sem que seja necessária a intervenção do usuário, otimizando a forma de acesso e recuperação às mensagens de correio eletrônico, aproveitando muitas das vantagens das soluções atuais, como a simplicidade do POP e a flexibilidade do IMAP, além de ser uma solução perfeitamente adaptável à estrutura atual do correio eletrônico Internet.

1.4. Organização do documento

O restante dessa dissertação está organizado da seguinte forma. No Capítulo 2 serão analisadas com mais detalhes as soluções já existentes para o acesso e recuperação de mensagens de correio eletrônico. No Capítulo 3 apresentaremos o protocolo *Smart Post Office Protocol* (SPOP) como uma alternativa para o acesso e recuperação de mensagens, enfocando alguns detalhes da sua especificação, seus comandos, bem como inovações propostas por este protocolo. No Capítulo 4 apresentaremos aspectos concernentes à implementação do protocolo SPOP, tecendo alguns comentários sobre o desenvolvimento tanto de servidores, quanto de clientes que implementem o protocolo SPOP.

No Capítulo 5 faremos uma exposição das conclusões atingidas com o trabalho, além da análise das perspectivas de trabalhos futuros. Refletiremos até onde conseguimos caminhar com este projeto e até onde este projeto poderá chegar.

Capítulo 2

Acesso e Recuperação de Mensagens de Correio Eletrônico Internet

Na origem do sistema de correio eletrônico as mensagens eram tipicamente recebidas e armazenadas em um único computador, que dentre outros nomes era chamado de servidor de correio eletrônico. Este servidor tinha os seus recursos compartilhados entre vários terminais. Todos os usuários podiam ter acesso a esta máquina, ou a um dos seus terminais, e manipular as suas mensagens eletrônicas. Mesmo sendo um modelo que continua sendo utilizado, ele possui duas principais limitações. A primeira é que ele não é escalável, no sentido de acomodar uma população crescente de usuários, pois além de abrigar o serviço de correio o sistema necessita atender aos usuários nas suas demais necessidades. A segunda limitação, está no fato de se estar preso a aplicações e funcionalidades próprias da plataforma em que se encontra o sistema de correio eletrônico. O primeiro ponto afeta a parte administrativa do sistema de correio eletrônico, e a segunda característica afeta diretamente a usabilidade deste sistema por parte dos usuários.

Para permitir a utilização de seus recursos por um grande número de terminais este servidor de correio necessitaria estar executando em grandes computadores. Nesta arquitetura identificamos que temos um ônus maior para termos um ambiente com interface gráfica para o usuário, comum em outras arquiteturas como no caso dos computadores pessoais. Outro prejuízo dessa arquitetura está no fato de não suportar a participação no sistema de correio de computadores pessoais ou portáteis, que ocasionalmente se conectam à rede, utilizando os mesmos programas que um usuário já está familiarizado com o seu uso.

Tornou-se necessário, desta forma, o desenvolvimento de uma arquitetura distribuída de troca de mensagens, além de novos paradigmas de acesso às mensagens que permitissem a participação de computadores pessoais no sistema de correio eletrônico. Contudo, antes de considerar protocolos de acesso e recuperação de mensagens (cliente-servidor) específicos para suportar essa nova arquitetura, é

importante conhecermos e entendermos bem os diferentes paradigmas para o acesso a mensagens de correio eletrônico.

2.1. Paradigmas de Acesso a Mensagens

O documento de padronização da Internet que define os paradigmas de acesso a mensagens é a RFC 1733. Este documento define três modos de operação ou paradigmas de acesso a mensagens em um sistema de troca de mensagens distribuído: *online*, *offline* e desconectado.

2.1.1. Modo de operação *online*

Na operação *online*, as mensagens permanecem no servidor de correio eletrônico e são manipuladas, a partir de uma outra máquina, por um programa cliente de correio eletrônico. O modo de operação *online* é um modelo genérico, que surgiu a partir do funcionamento do acesso a mensagens quando o servidor e o cliente de correio localizavam-se fisicamente na mesma máquina. Este modo de operação permite que as mensagens de uma caixa de correio sejam acessadas e manipuladas remotamente, a partir de diferentes localidades em diferentes instantes. Este modo de operação não está vinculado a um sistema de arquivos em particular, como acontecia antes de se ter protocolos específicos para o acesso a mensagens. O modo de operação *online* foi desenvolvido especificamente para máquinas conectadas permanentemente à Internet.

2.1.2. Modo de operação *offline*

Devido ao surgimento da tecnologia que permite que computadores pessoais temporariamente se conectem à rede, desenvolveu-se o modelo *offline* de acesso a mensagens que permite a estes computadores, excluídos do modelo inicial, terem acesso ao serviço de troca mensagens.

Na operação *offline*, ou modo de operação em lote (*batch*), as mensagens são armazenadas temporariamente em um servidor, e recuperadas por clientes de correio eletrônico localizadas em diferentes máquinas. O UA do usuário, a partir de uma estação de trabalho ou um computador pessoal, periodicamente conecta-se ao servidor e move todas as suas mensagens para a máquina local. Uma vez transferidas as mensagens, elas são manipuladas localmente na máquina do cliente.

A operação *offline* pode ser melhor compreendida pensando-se no acesso a mensagens como sendo o último passo no processo de armazenamento e envio do sistema de correio eletrônico, processo conhecido como *store-and-forward*. Este último

passo, que é a transferência das mensagens, é iniciado pelo usuário através do programa cliente de correio.

O modelo *offline* não é apropriado se alguém necessita ter acesso ao depósito das suas mensagens a partir de diversas máquinas, em diferentes instantes. Por exemplo, o modelo *offline* impede que um usuário manipule as suas mensagens de correio eletrônico a partir de uma máquina diferente daquela em que é mantido o depósito de suas mensagens. O depósito de mensagens está geralmente vinculado a um cliente de correio eletrônico em particular, e desenvolvido para uso local.

2.1.3. Modo de operação desconectado

No modelo de operação desconectado, um programa cliente conecta-se ao servidor de acesso a mensagens de correio eletrônico, copia de forma temporária, algumas mensagens selecionadas pelo usuário, da área do servidor para o disco local da máquina onde está o cliente, e então desconecta-se do servidor. Uma vez que as mensagens estejam na área local do usuário, este pode operar com as cópias locais das mensagens, para posteriormente voltar a se conectar com o servidor, e ressincronizar o estado das cópias locais das suas mensagens com a sua caixa de correio armazenada no servidor. Este modelo se diferencia do modelo *offline* puro, porque no modelo desconectado as cópias originais das mensagens permanecem no servidor. O programa cliente de correio transfere as mensagens que o usuário selecionou para a sua área local, e deve fazer posteriormente uma nova conexão com o servidor para ressincronizar o estado de cada mensagem que foi transferida para a área local.

As operações *online* e desconectada são compatíveis, e um cliente pode muito bem alternar a sua operação entre uma e outra. No entanto, nenhuma das duas é compatível com o modelo *offline*, uma vez que, por definição, a operação *offline* implica em descartar a mensagem no servidor depois que ela tenha sido copiada para o disco local da máquina do cliente.

Uma crescente fração dos usuários de correio eletrônico utiliza uma máquina no trabalho e uma outra em sua casa, e possivelmente um computador portátil para as suas viagens. Este perfil de usuários pode preferir, se lhes for dada a escolha, os modelos de acesso *online* ou desconectado, pois são os únicos que possibilitam, na prática, uma solução distribuída. A opção pelo acesso *offline* apenas é justificada quando, para acessar as suas mensagens, um usuário pretende utilizar uma única máquina, ou o tempo de conexão e os recursos do servidor possuírem um custo elevado.

Enquanto o acesso *online* implica em longos períodos de conexão com o servidor, o modelo de acesso desconectado requer pouco mais que o tempo de conexão que o modelo de acesso *offline*. A preferência geral pelo modelo *offline* não pode ser inferida a partir da sua larga presença no mercado, pois para a maior parte desses usuários nunca foi oferecida uma outra opção.

As definições de paradigmas que vimos são implementadas pelos protocolos de acesso a mensagens. Veremos em seguida a arquitetura dos protocolos de acesso a mensagens mais utilizados no momento. Não existe uma correspondência direta entre um protocolo de acesso a mensagens e um destes paradigmas de acesso, visto que um mesmo protocolo pode implementar mais de um deles.

2.2. Estrutura de uma Mensagem Eletrônica

No decorrer deste documento faremos referência à estrutura de uma mensagem de correio eletrônico bem como a suas partes. Nesta seção faremos uma exposição da estrutura de mensagens de correio eletrônico definida pela RFC 822.

Após o desenvolvimento de diversos padrões para mensagens de texto enviadas entre computadores, surgiu a necessidade de se desenvolver um formato padrão para estas mensagens de texto, que fosse independente do ambiente, de forma que pudesse ser aplicada a outros sistemas de troca de mensagens via rede. Este padrão foi publicado através da RFC 822, que embora tenha sido desenvolvido pela mesma equipe que desenvolveu o SMTP foi publicado em separado por se tratar de uma proposta genérica passível de aplicação em outros sistemas.

A RFC 822 define um conjunto de regras sintáticas para mensagens de texto que são referenciadas dentro da descrição do protocolo SMTP (RFC 821). Um programa cliente do sistema de correio eletrônico Internet tem de estar habilitado a criar mensagens de texto no formato definido pela RFC 822 e enviar esta mensagem para um receptor SMTP (geralmente um servidor de correio eletrônico Internet) de acordo com as normas definidas pela RFC 821.

De acordo com as regras definidas pela RFC 822, uma mensagem é dividida em duas partes: envelope e conteúdo. O envelope contém todas as informações necessárias para que sejam completadas a transmissão e entrega da mensagem. O conteúdo é composto pelo objeto a ser entregue ao destinatário. Na maioria dos casos o corpo de uma mensagem eletrônica varia entre um conjunto linhas de texto sem formatação ou mensagens no formato MIME, o qual permite inserir no corpo de uma mensagem elementos de vários tipos.

possuem definições claras sobre os tipos de arquivos, e portanto torna-se impossível para protocolos genéricos determinar quando aplicar convenções predeterminadas. Assim, o programa cliente de correio eletrônico estará amarrado ao formato específico de armazenamento do servidor.

Outra desvantagem está no sistema de proteção dos arquivos e questões de segurança, que têm provado ser problemáticas em alguns protocolos de acesso a arquivos. O grau de confiança num protocolo de acesso a arquivos pode levar ao uso ineficiente da capacidade de transmissão da rede, quando arquivos completos são transferidos várias vezes através da rede para se efetuar a verificação da integridade, onde podem ser determinadas a presença ou ausência de trechos particulares destes arquivos.

Quando estamos tratando com protocolos específicos para a recuperação e o acesso a mensagens de correio eletrônico, podemos pensar em ajustes na utilização destes protocolos para maximizar o desempenho dentro do domínio de uma aplicação específica. Podemos também, permitir uma distribuição lógica do processamento entre o cliente e o servidor, criando clientes inteligentes que realizem algumas tarefas sem ter que fazer pedidos ao servidor, minimizando assim os dados transmitidos através da rede entre o cliente e o servidor.

Os protocolos específicos para o acesso e recuperação de mensagens, diferentemente dos protocolos de sistemas de arquivos, podem ser instalados, geralmente, sem nenhum privilégio especial, trazendo mais segurança para o sistema, além de ocultar do programa cliente o formato de armazenamento das mensagens utilizado pelo servidor.

Desta forma, ao se pensar em desenvolver uma nova solução para o acesso e recuperação de mensagens de correio eletrônico, a utilização de um protocolo específico de acesso a mensagens torna-se a opção mais indicada. O que vemos hoje, são os protocolos TCP e IP como um padrão de fato no que diz respeito à tecnologia de rede de computadores, e é baseado neste fato que se ressalta mais uma vez a opção de trabalhar uma solução voltada para esta tecnologia. No decorrer do documento, quando quisermos nos referir ao padrão estipulado pelos protocolos TCP e IP, utilizaremos o termo Internet. Os protocolos de acesso e recuperação de mensagens utilizado na Internet são: o POP, o IMAP (já definidos anteriormente) e o DMSP (*Distributed Mail System Protocol*) [Clark86] [Clark86-2] [Lambe88].

Destes três, o mais antigo e mais conhecido é o POP, o qual foi originalmente definido pela RFC 918 [Reyno84] de outubro de 1984. Da mesma forma que os outros

mensagens, também se responsabiliza pelo armazenamento das mensagens dos usuários associados a ela. Desta forma as mensagens são entregues e armazenadas em um computador mas são manipuladas efetivamente a partir de um outro. Para que isso ocorra, torna-se necessário um protocolo que acesse ou recupere as mensagens no servidor de correio eletrônico.

Primeiramente, para a configuração de um sistema de correio eletrônico são tomadas decisões que levam em consideração aspectos de segurança e de administração do sistema. Como decisão administrativa, podemos citar a escolha do local onde serão armazenadas as mensagens eletrônicas de um determinado usuário, e se este armazenamento será temporário ou permanente. Quanto ao local de armazenamento, podem ser utilizado o computador pessoal do usuário, um servidor central da organização à qual o usuário pertence; ou as mensagens de uma organização ainda podem ser armazenadas em vários servidores departamentais. Após definidas todas as políticas administrativas do sistema de correio eletrônico, torna-se necessário estabelecer qual será a forma de acesso às mensagens e qual o protocolo que será utilizado.

Na escolha desses protocolos é possível optar por protocolos genéricos de acesso a sistemas de arquivos distribuídos, como o NFS, ou por protocolos específicos para o acesso e recuperação de mensagens de correio eletrônico, como POP ou IMAP.

Vamos analisar a opção de termos um protocolo de sistema de arquivos para ter acesso ao que podemos chamar de depósito remoto de mensagens. Surgem alguns problemas decorrentes desta estratégia que devem ser levados em consideração. Em primeiro lugar, tem-se a dificuldade de se integrar computadores de diferentes arquiteturas através de um mesmo protocolo de sistema de arquivos, o que aumenta bastante o custo para se ter computadores de diferentes plataformas participando de um mesmo sistema de correio eletrônico. Em segundo lugar, os protocolos de sistema de arquivos exigem muitos recursos do sistema operacional e da própria rede. Precisa-se uma LAN[1] rápida para viabilizar respostas rápidas aos usuários abstraindo a infra-estrutura de rede que está sendo utilizada.

A maioria dos protocolos de sistema de arquivos não pode mapear caracteres especiais como quebra de linha, da mesma forma que não pode mapear as diferenças de convenções entre sistemas operacionais. Os sistemas de arquivos também não

[1] LAN é a abreviatura de *Local Área Network*, no texto ela designa qualquer tipo de rede local de computadores. Ou seja, uma rede de computadores com área de cobertura limitada.

eles: ***Resent-Message-ID***, ***In-Reply-To***, ***References***, ***Keywords***, ***Comments***, ***Encrypted***, ***User-Defined-Field*** e outros.

A assinatura de uma mensagem, que é a sua identificação universal, é atribuída automaticamente às mensagens quando estas são enviadas a partir de um cliente de correio eletrônico qualquer. O formato para a assinatura de uma mensagem também é definido pela RFC 822. Segue abaixo o exemplo do código fonte de uma mensagem compatível com a RFC 822.

```
Received: from dsc.ufpb.br [150.165.75.21] by lsd.dsc.ufpb.br [150.165.85.5]
with SMTP (MDaemon.v2.7.SP5.R) for <fellipe@lsd.dsc.ufpb.br>; Fri, 02 Jul
1999 23:12:11 -0300
Received: from localhost (fellipe@localhost)
        by dsc.ufpb.br (8.8.5/8.8.5) with SMTP id XAA32638;
        Fri, 2 Jul 1999 23:05:37 -0300
Date: Fri, 2 Jul 1999 23:05:37 -0300 (GRNLNDST)
From: "Fellipe Araujo Aleixo / (Professor DSC)" <fellipe@dsc.ufpb.br>
X-Sender: fellipe@anjinho
To: fellipe@lsd.dsc.ufpb.br
cc: Minha conta na OpenLine <aleixo@openline.com.br>
Subject: Mensagem compatível com a RFC821
Message-ID: <Pine.A41.3.95.990702230117.30682A-100000@anjinho>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=iso-8859-9
Content-Transfer-Encoding: QUOTED-PRINTABLE
X-MDaemon-Deliver-To: fellipe@lsd.dsc.ufpb.br
X-Return-Path: fellipe@dsc.ufpb.br



Esta mensagem é um exemplo de uma mensagem compatível com o formato
definido pela RFC 821. É importante lembrar que mesmo tendo sido
definida conjuntamente com a RFC 821 que define o sistema de correio
eletrônico Internet, através do protocolo SMTP, a definição da RFC é uma
definição geral e pode ser utilizada por outros sistemas.


----
Fellipe Aleixo
Departamento de Sistemas e Computação
Centro de Ciências e Tecnologia
Universidade Federal da Paraíba
```

Figura 2.1 – Código fonte de uma mensagem compatível com a RFC 822

2.3. Protocolos de Acesso a Mensagens

Atualmente, a maioria dos usuários de sistemas de acesso e recuperação de mensagens de correio eletrônico não têm suas mensagens entregues diretamente no seu computador pessoal. As mensagens enviadas para o usuário são armazenadas em um servidor de correio eletrônico que executa em um computador mais confiável, por exemplo o servidor de uma organização ou de um provedor de acesso que estejam permanentemente ligados à Internet. Este servidor de correio eletrônico geralmente serve a todos os usuários de determinada organização com o serviço de caixa de correio. O serviço de caixa de correio é a denominação dada para o serviço prestado por uma máquina que além de ser responsável pela transferência das

O código fonte de uma mensagem consiste em campos de cabeçalho e opcionalmente um corpo. O cabeçalho é separado do corpo por uma linha em branco. Cada campo do cabeçalho pode ser visto como uma única linha lógica de caracteres ASCII, composta pelo nome do campo e o valor do campo.

Os campos não aparecem em uma ordem particular, apenas poucos são necessários, mas existem muitos campos opcionais. Apenas alguns dos campos de cabeçalho de uma mensagem são úteis ao usuário. Na maioria dos casos estes campos não são exibidos pelo programa cliente de correio eletrônico; são exibidos apenas alguns campos mais significativos que identificam a mensagem.

Os campos de cabeçalho que são obrigatórios são os seguintes: ***From***, que indica quem enviou a referida mensagem; ***Date***, que indica a data e o instante de tempo em que a mensagem foi enviada; ***To***, que indica o endereço ou lista de endereços, separados por vírgula, para os quais a mensagem foi enviada; ***CC***, que indica um endereço ou lista de endereços para os quais será enviada uma cópia ("cópia carbono") da referida mensagem; ***Subject***, que contém uma descrição resumida do assunto da mensagem.

Existem muitos outros campos de cabeçalho de uso opcional, destacaremos alguns. ***Received***, que indica as informações a respeito do recebimento da mensagem como o domínio de origem, a rota, o protocolo, um identificador da mensagem, além das informações da data e instante de tempo do recebimento. Este campo de cabeçalho gerado automaticamente quando esta percorre o sistema de transferência de mensagens, onde um novo campo ***Received*** é gerado quando a mensagem é recebida em um MTA. ***Sender***, que é utilizado quando a pessoa que esta enviando a mensagem não é a mesma que deu origem à mesma. ***Reply-To***, que indica o endereço que deverá ser utilizado para eventuais respostas a uma referida mensagem (se este campo não estiver presente será usado o endereço do campo ***From***). ***Return-Path***, que tem a mesma função que o campo ***Reply-To*** só que acrescenta uma rota opcional. ***Message-ID***, que armazena a identidade universal da mensagem. A identidade de uma mensagem é um campo gerado no ato da criação da mensagem pelo UA (ou pelo próprio sistema de transferência de mensagens se o UA não o fizer). Esta identificação segue um formato padrão e é construída no sentido de que cada identidade gerada seja diferente uma da outra, por isso é chamada de universal.

Os demais campos de cabeçalho definidos, são utilizados em casos especiais, para prover maiores informações aos programas clientes de correio eletrônico, são

dois, o POP passou por muitas revisões desde a sua criação. A última versão do POP é a de número 3, que está descrita na RFC 1725. Nesta revisão estão inclusos o suporte para métodos avançados de autenticação e identificadores únicos para mensagens, ambos derivados do trabalho desenvolvido com o IMAP versão 4. O POP foi desenvolvido especificamente para suportar o paradigma *offline* de acesso a mensagens. Contudo, de forma adaptada e com muitas limitações, também pode ser aplicado no suporte aos outros dois paradigmas de acesso a mensagens.

O DMSP foi definido pela primeira vez na RFC 984 [Clark86] de maio de 1986. A sua mais recente revisão esta na RFC 1056 de junho de 1988. Diferentemente do POP, o DMSP não é amplamente suportado, pois é restrito a uma única aplicação, o PCMAIL [Clark86] [Clark86-2] [Lambe88]. O DMSP foi desenvolvido especificamente para dar suporte ao modelo de operação desconectado do PCMAIL.

O IMAP foi desenvolvido em 1986 na Universidade de Stanford, mas não foi documentado até a RFC 1064 de julho de 1988. Ele também passou por muitas revisões desde então, e tem como versão mais recente a primeira revisão da versão 4, descrita na RFC 2060. O IMAP foi originalmente desenvolvido para suportar o modelo de acesso *online*; por esta razão o IMAP no seu início se chamou *Interactive Mail Access Protocol*. O nome mudou em 1993 para *Internet Message Access Protocol* (continuando com a mesma sigla) como parte do esforço de padronização do IETF para melhor representar as capacidades atuais do protocolo. O IMAP é um superconjunto funcional das capacidades do POP que pode suportar completamente o acesso *offline* da mesma forma que o POP. Com os acréscimos feitos na versão 4 ele passou a suportar a operação desconectada. Por isso o IMAP englobou as funcionalidades de ambos POP e DMSP.

No anexo A deste documento você encontra uma ilustração do processo de envio e recebimento de mensagens no correio eletrônico Internet. A seguir, discutiremos os dois principais protocolos de acesso a mensagens, quais sejam: o POP e o IMAP.

2.3.1. POP: Post Office Protocol

Antes de iniciarmos uma descrição mais detalhada do POP, ressaltamos que ele é o protocolo mais popularmente utilizado para acessar e recuperar mensagens de correio eletrônico na Internet. Mesmo com as previsões que indicam que em pouco tempo será superado pelo IMAP [Hughe98], o POP continuará a ser suportado por um bom tempo, visto a grande base instalada que utiliza o mesmo. O POP implementa basicamente o modelo *offline* de acesso a mensagens.

A primeira edição do POP se deu através da RFC 918, de outubro de 1984, chegou a sua segunda versão (POP2) em fevereiro de 1985, através da RFC 937 [Butle85], e atingiu a sua versão atual, que é a terceira (POP3), em novembro de 1988, através da RFC 1081 [Rose88]. O POP3 foi revisado em maio de 1991 (RFC 1225) [Rose91], junho de 1993 (RFC 1460) [Rose93], novembro de 1994 (RFC 1725) [Myers94] e por fim em maio de 1996 (RFC 1939) [Myers96]. Trataremos aqui da última versão do POP.

A proposta do POP baseia-se no cenário em que existe uma máquina mais simples, utilizada pelo usuário, e uma máquina com mais recursos que possua um servidor de correio eletrônico oferecendo o serviço de armazenamento temporário de mensagens, conhecido como serviço de caixa de correio. Neste cenário, o POP é a ferramenta utilizada para efetuar o transporte de mensagens do servidor de caixa de correio para a máquina do usuário.

O POP é um protocolo baseado em comandos e respostas, e é implementado sobre um protocolo de transporte confiável, geralmente o TCP. A comunicação entre cliente e servidor acontece em ASCII e é orientada a linhas. O servidor POP envia uma saudação em resposta a uma nova conexão estabelecida. Então, o cliente repete o ciclo de enviar um comando e esperar pela resposta correspondente. Na Figura 2.2 vemos uma ilustração dos estados de uma conexão entre um cliente e um servidor POP.

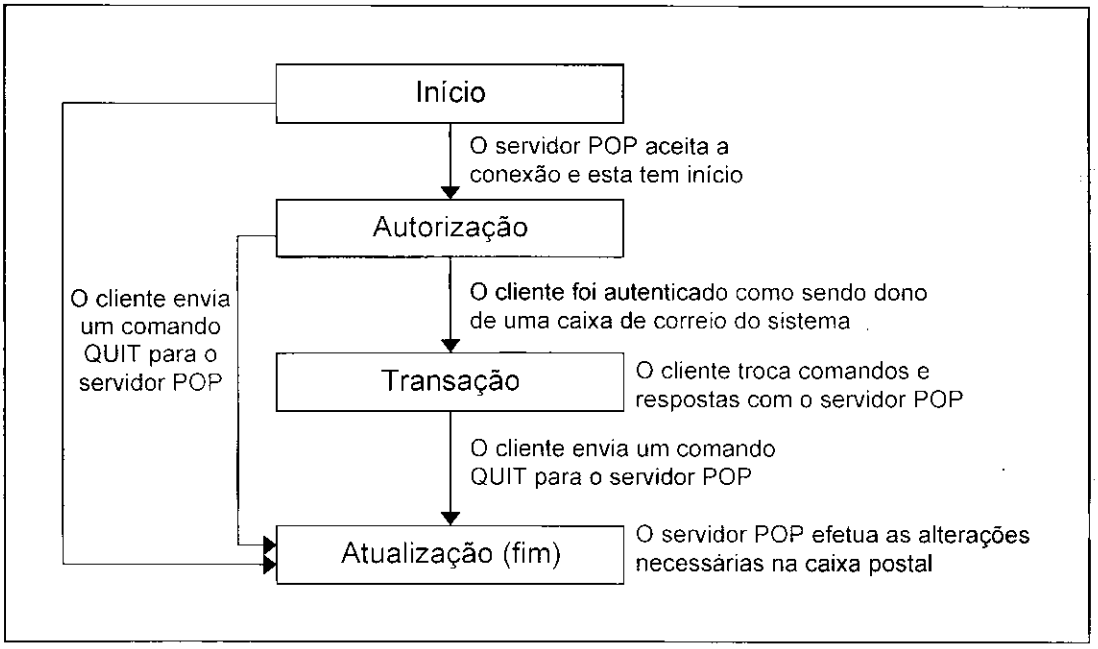

Figura 2.2 – Estados de uma conexão entre um cliente e um servidor POP.

Cada conexão com um servidor POP inicia no estado de "autorização". Neste estado o cliente só está apto a autenticar-se ou encerrar a conexão. Ao ter acesso ao estado de "transação" o cliente ganha o acesso exclusivo para manipulação com a caixa de correio. Por fim, quando o cliente enviar um comando QUIT o servidor remove todas as mensagens marcadas para serem apagadas, a caixa de correio é fechada e a conexão de acesso exclusivo à caixa de correio é revogada. Por fim, a conexão de rede com o cliente é encerrada. Se o cliente encerrar uma conexão sem um comando QUIT o servidor não pode entrar no estado de atualização.

Cada comando e resposta consiste em uma seqüência de caracteres ASCII seguidos por uma indicação de mudança de linha (*carriage return* e *line feed*). Cada comando inicia-se por tokens de quatro caracteres, seguidos por argumentos opcionais. As respostas consistem em uma linha de texto iniciada com um indicador de sucesso ou falha, seguida pelos dados que compõem a resposta do servidor. No POP a sequência "+OK" representa o sucesso e a sequência "–ERR" representa uma falha. Algumas das respostas do servidor POP podem ocupar várias linhas, terminadas por uma linha que contém um único ponto final.

O servidor POP é capaz de tratar um grande número de conexões simultâneas com UAs. Cada conexão requer acesso exclusivo à caixa de correio do usuário autenticado, com permissões de escrita e leitura.

O protocolo POP é um protocolo simples, possuindo poucos comandos, de relativa simplicidade de implementação. A Tabela 2.1 apresenta uma breve descrição dos comandos oferecidos. Basicamente, o POP possibilita a transferência e posterior exclusão de mensagens do servidor de caixa de correio, atuando como interface entre o UA e o MS. Entre os objetivos do POP não está a disponibilização de operações para a manipulação das mensagens, o que torna este protocolo bastante limitado. Entretanto, como conseqüência desta limitação, obtém-se a vantagem de se ter um MS mais simples e consequentemente mais rápido.

Uma conexão típica do POP segue os seguintes passos: o administrador dá início ao servidor POP, que passa a esperar requisições de conexão destinados à porta TCP 110 da máquina que o hospeda. Um usuário, através de um UA, deseja fazer uso do serviço, e tenta estabelecer uma conexão TCP com o servidor, enviando uma mensagem de requisição de conexão para a porta 110. O servidor, estando disponível, aceita a conexão e envia uma mensagem de saudação. O primeiro passo depois da conexão estabelecida é a autenticação de cliente. Após a autenticação, o cliente verifica a existência de novas mensagens e efetua a recuperação destas novas

mensagens, que estão armazenadas no servidor. Por fim, o cliente instrui o servidor a excluir todas as mensagens que foram transferidas, e encerra a conexão.

Estado da conexão	Comando	Breve descrição
Estado de Autorização	USER	Inicia a autenticação em modo texto, e tem como argumento o nome que representa a caixa de correio do usuário.
	PASS	Segue ao comando USER, e seu argumento é a senha de acesso para a caixa de correio informada anteriormente.
	APOP	Efetua a autenticação sem que a senha do usuário seja enviada em modo texto, utilizando-se um algoritmo de sumarização (como o MD5).
	AUTH	Este comando possibilita que a autenticação seja feita através de mecanismos avançados de criptografia, como o Kerberos [Kohl93].
Estado de Transação	STAT	Recupera o estado (número de mensagens armazenadas e o total em bytes destas mensagens) da caixa de correio.
	LIST	Recupera as informações de uma mensagem específica ou de todas as mensagens de uma caixa de correio.
	UIDL	Retorna o identificador único de mensagens para uma mensagem específica ou de todas as mensagens de uma caixa de correio.
	RETR	Efetua a recuperação de uma mensagem específica.
	TOP	Recupera um determinado número de linhas do cabeçalho de uma mensagem específica.
	DELE	Marca uma referida mensagem para ser excluída.
	NOOP	Não requer nenhum atividade, apenas uma resposta positiva.
	RSET	Retira a marca para exclusão de todas as mensagens.
	QUIT	Encerra a transação e dá inicio ao estado de atualização.
Estado de Atualização		Neste estado não são permitidos comandos. Apenas as mensagens previamente marcadas para exclusão são permanentemente removidas, a caixa de correio é fechada, e o acesso exclusivo é liberado.

Tabela 2.1 – Breve descrição dos comandos do POP (versão 3).

Dentre as limitações do POP, destacaremos algumas que consideramos importantes, pois dificultam a manipulação das mensagens por parte do usuário. O POP não possui outras funcionalidades, a não ser efetuar a transferência das mensagens de uma caixa de correio, situada no servidor de correio, para a máquina local. O fato é que, não importa se a caixa de correio está com muitas mensagens, o UA se conectará ao servidor POP e iniciará o transporte de todas as mensagens, resultando em uma operação que consumirá uma grande quantidade de tempo, visto o volume de mensagens que deverá ser transferido. O problema é melhor percebido quando o número de usuários suportados pelo servidor de correio também for muito grande. Nesse caso só o tráfego gerado entre um grande número de UAs executando simultaneamente e o servidor POP, congestionará a rede e sobrecarregará a máquina onde está localizado o servidor de correio.

Um outro ponto negativo está no fato do POP não trabalhar com pastas de arquivamento no servidor. Desta forma o usuário não pode classificar suas mensagens em pastas, visando facilitar a sua manipulação, pois o POP só reconhece as mensagens que estiverem na caixa de entrada e só efetuará o transporte destas. Neste ponto o POP é incompatível com a solução IMAP, que é a única solução padronizada, até o momento, a reconhecer a estrutura de pastas.

Outro problema, é que o servidor POP não tem como enviar uma mensagem que chega após o início da transferência das mensagens de uma determinada caixa postal eletrônica, pois é o UA que tem de especificar quais as mensagens que têm que ser transferidas. O primeiro procedimento do UA ao se conectar com um servidor POP é obter a lista das mensagens da caixa de correio, e só de posse desta informação passar a requisitá-las uma a uma. Mas se uma mensagem chegar durante a transferência, ela não vai ser enviada, a não ser que o UA após acabar a transferência das mensagens que havia requisitado anteriormente, peça uma nova lista das mensagens que estão na caixa de entrada do usuário, e havendo novas mensagens, ele venha a requisitar a recuperação destas mensagens recém chegadas. Portanto, a responsabilidade pela transferência ou não das mensagens recém chegadas é unicamente do UA.

Alguns UAs baseados na solução POP possuem a opção para deixar as mensagens no servidor. Essa opção pode ser interessante para usuários com a necessidade de acessar suas caixas postais a partir de vários locais. Para possibilitar esta opção o cliente POP não utiliza o comando DELE para apagar as mensagens, e utiliza o comando UIDL no início de cada conexão para determinar quais mensagens no servidor foram recuperadas previamente, para um dado MS local. Trabalhando-se com o identificador universal de mensagens, efetua-se apenas a exclusão lógica de mensagens, visto que as mensagens permanecem no servidor.

Esta opção acarreta algumas sérias deficiências no sistema como um todo. Não excluindo as mensagens, corre-se o perigo da superlotação do servidor (MS). O acúmulo de mensagens no servidor leva, por exemplo, à implantação de um sistema de quotas para as caixas de correio, ou até mesmo em políticas de exclusão de mensagens que já foram lidas ou que ultrapassarem um período de residência máximo tolerado. Um esquema típico faz com que mensagens mais velhas que 30 dias sejam excluídas do servidor.

Se um UA, de alguma forma, perder a lista de UIDLs das mensagens recuperadas anteriormente, a única escolha será recuperar todas as mensagens que

permanecem no MS do servidor, tendo sido recuperadas anteriormente ou não. Isto pode resultar em uma recuperação muito longa, incluindo mensagens anteriormente excluídas, que podem inclusive conter arquivos anexados, além de duplicatas de mensagens locais.

O nível de segurança do POP protege as mensagens requerendo autenticação do usuário antes de prover o acesso para as caixas de correio. A forma de autenticação mais comum adotada pelos clientes POP é feita enviando-se, através da rede, o nome do usuário da caixa de correio e a senha de acesso correspondente, ambos em formato texto. Essa estratégia é extremamente insegura, pois já existem hoje várias formas de capturar pacotes que trafegam na rede e ter acesso aos seus conteúdos. Desta forma, seria relativamente fácil alguém conseguir capturar as informações de acesso a uma caixa de correio, então ter acesso às mensagens particulares daquela caixa de correio.

2.3.2. IMAP: Internet Message Access Protocol

O IMAP veio como uma proposta para modificar a arquitetura de serviço existente, eliminando as deficiências do POP. A proposta do IMAP é ser um superconjunto do POP. Para tal o IMAP realiza todas as operações oferecidas pelo POP. A idéia do IMAP é proporcionar que estações de trabalho, computadores pessoais e outros tipos de máquinas com menos recursos tenham acesso a mensagens eletrônicas armazenadas em um servidor central de correio eletrônico. O IMAP surgiu como uma solução para integrar essas máquinas, possivelmente de diferentes plataformas, em um mesmo sistema de correio eletrônico distribuído. A sua proposta baseia-se em centralizar o serviço de troca e armazenamento de mensagens, como meio de facilitar a administração.

A primeira versão do IMAP a se tornar padrão foi publicada em julho de 1985, através da RFC 1064 (IMAP2) [Crisp88]. A publicação da versão 4 (IMAP4) ocorreu em dezembro de 1994, através das RFCs 1730, 1731, 1732 e 1733. A primeira revisão da versão 4 (IMAP4 rev.1) foi publicada em 1996, através das RFCs 2060 [Crisp96-3], 2061 [Crisp96-2] e 2062 [Crisp96].

Os dois principais requisitos atingidos com o desenvolvimento do IMAP foram a necessidade de acessar mensagens de correio eletrônico a partir de diversos pontos da rede e a necessidade de diminuir a utilização da banda passante na conexão entre o UA e o MS. A sua proposta fundamental concentra-se em manter as mensagens no servidor e possibilitar um meio capaz de efetuar a recuperação seletiva e o gerenciamento das mensagens no servidor.

O IMAP oferece o suporte para que as mensagens do usuário sejam organizadas em múltiplas pastas na caixa de correio do usuário, para possibilitar que elas sejam acessadas e administradas a partir de vários pontos da rede. O IMAP permite também, uma recuperação de informação e mensagens muito mais sofisticada que o POP.

Mantendo múltiplas pastas na caixa de correio de um usuário, o IMAP reproduz a maneira que muitos UAs baseados no POP utilizam para gerenciar o seu MS local. Essas pastas podem ser criadas e manipuladas pelo usuário, através dos UAs que implementam o IMAP, podendo ser organizadas de várias formas, como por exemplo hierarquicamente. Uma vantagem desta estruturação no servidor, é que qualquer cliente IMAP utilizado pelo usuário em questão vai ter acesso à mesma organização da caixa de correio, não importando a partir de onde está sendo usado, contanto que esteja acessando o mesmo servidor IMAP.

O IMAP também provê uma recuperação seletiva de mensagens. Esta seletividade tanto pode ser em termos de que partes de uma mensagem se deseja recuperar, ou que mensagens se deseja recuperar, visto que o IMAP também provê uma maneira de efetuar buscas na caixa postal de um usuário utilizando vários critérios. Essa facilidade permite que sejam feitas buscas mais sofisticadas nas caixas de correio dos usuários sem aumentar a complexidade dos UAs utilizados, nem requerendo que o computador do usuário possua uma capacidade de processamento relativamente alta.

A pesquisa remota baseada no servidor (*remote searching*), que possibilita a busca por padrões de texto nas mensagens contidas em uma determinada caixa de correio, é implementada pelo comando SEARCH. A grande vantagem da pesquisa baseada no servidor é que o usuário não precisa transferir todo o conteúdo de uma caixa de correio para a sua máquina local, para só então efetuar uma busca nas suas mensagens por um determinado padrão de texto. Com a pesquisa remota, apenas o comando SEARCH e o seu resultado são transmitidos através da rede, fato esse que acarreta uma redução do tráfego gerado.

A construção de critérios para o comando SEARCH se dá com a utilização das chaves de busca mostradas na Tabela 2.2. Podem ser especificados múltiplos critérios para a busca de mensagens realizada pelo comando SEARCH. Neste caso temos chaves de busca separadas por espaços em branco. O resultado é a interseção das mensagens que satisfizerem todos os critérios utilizados, eqüivalendo à operação lógica "e" (*and*).

Chave de busca	Descrição
[número da mensagem]	Seleciona a mensagem com o respectivo número de sequência
ALL	Seleciona todas as mensagens
ANSWERED	Seleciona mensagens com o sinalizador "\Answered" (respondida)
BCC [texto]	Seleciona mensagens que contêm o "texto" em questão no campo BCC
BEFORE [data de envio]	Seleciona mensagens com a data interna menor que a data informada
BODY [texto]	Seleciona mensagens que contêm o "texto" em questão no campo BODY
CC [texto]	Seleciona mensagens que contêm o "texto" em questão no campo CC
DELETED	Seleciona mensagens com o sinalizador "\Deleted" (excluída)
DRAFT	Seleciona mensagens com o sinalizador "\Draft" (rascunho)
FLAGGED	Seleciona mensagens com o sinalizador "\Flagged" (sinalizada)
FROM [texto]	Seleciona mensagens que contêm o "texto" em questão no campo FROM
HEADER [campo] [texto]	Seleciona mensagens que contêm o "texto" em questão no "campo" especificado (From, To, etc.)
KEYWORD [sinalizador]	Seleciona mensagens com o "sinalizador" especificado
LARGER [tamanho]	Seleciona mensagens que possuam tamanho maior que o número de bytes especificado
NEW	Seleciona mensagens que tenham o sinalizador "\Recent" (recente) mas não possuam o sinalizador "\Seen" (lida); é equivalente à construção "RECENT UNSEEN"
NOT [chave de busca]	Seleciona mensagens que não satisfizeram ao critério especificado
OLD	Seleciona mensagens que não possuem o sinalizador "\Recent" (recente), equivalente a "NOT RECENT"
ON [data]	Seleciona mensagens com a data interna especificada
OR [chave de busca 1] [chave de busca 2]	Seleciona mensagens que satisfaçam um dos dois critérios de busca especificados, ou ambos
RECENT	Seleciona mensagens com o sinalizador "\Recent" (recente)
SEEN	Seleciona mensagens com o sinalizador "\Seen" (lida)
SENTBEFORE [data]	Seleciona mensagens com a data de envio menor que a "data" especificada
SENTON [data]	Seleciona mensagens com a data de envio igual à "data" especificada
SENTSINCE [data]	Seleciona mensagens com a data de envio maior ou igual à "data" especificada
SMALLER [tamanho]	Seleciona mensagens com tamanho menor do que o número de bytes especificado
SUBJECT [texto]	Seleciona mensagens que contêm o "texto" em questão no campo SUBJECT
TEXT [texto]	Seleciona mensagens que contêm o "texto" em questão em qualquer campo do cabeçalho ou no corpo da mesma
TO [texto]	Seleciona mensagens que contêm o "texto" em questão no campo TO
UID [UID da mensagem]	Seleciona uma mensagem com o UID especificado
UNASWERED	Seleciona mensagens que não possuam o sinalizador "\Answered"
UNDLETED	Seleciona mensagens que não possuam o sinalizador "\Deleted"
UNDRAFT	Seleciona mensagens que não possuam o sinalizador "\Draft"
UNFLAGGED	Seleciona mensagens que não possuam o sinalizador "\Flagged"
UNKEYWORD [sinalizador]	Seleciona mensagens que não possuam o "sinalizador" especificado
UNSEEN	Seleciona mensagens que não possuam o sinalizador "\Seen"

Tabela 2.2 – Componentes para criação de critérios de busca.

Quando efetua-se buscas em uma caixa de correio local não é gerado tráfego de rede, mas leva cada UA utilizado por um dado usuário a manter uma cópia local completa e atualizada da caixa de correio que está no MS. Além de que o UA tem que embutir dentro de si a complexidade da funcionalidade de pesquisa a mensagens. O

lado negativo é que a pesquisa impõe uma sobrecarga extra à máquina que executa o servidor IMAP. Isso exige máquinas com maiores capacidades de processamento e maiores taxas de entrada e saída. Por esta razão, uma mesma plataforma suporta menos usuários IMAP que usuários POP.

O IMAP mantém bastante informações sobre o estado das mensagens (se foi lida, respondida, etc.). Este fato, permite que os clientes IMAP possam prover mais informações sobre o estado das mensagens do que o usuário de sistemas proprietários tem geralmente a sua disposição. Essa carga de informação adicional sobre o estado das mensagens, que torna possível algumas funcionalidades do IMAP, torna o MS do servidor de correio mais complexo e mais lento do que um que atenda exclusivamente para POP.

Com o IMAP a autenticação dos usuários ganhou mais segurança com a adição de mecanismos internos ao protocolo, como esquemas de criptografia, como por exemplo o Kerberos [Kohl93]. Este tipo de autenticação no POP era feito por módulos externos não definidos claramente.

Além disso o IMAP foi desenvolvido para trabalhar bem com implementações assíncronas e utilizando a tecnologia de *pipelining*. Para prover o *pipelining* o IMAP oferece o conceito de respostas rotuladas (*tagged responses*) no diálogo entre o servidor e o cliente. Cada comando que o cliente envia ao servidor pode conter um rótulo, e este comando não é considerado satisfeito até que seja recebida uma resposta com o mesmo rótulo. A utilização de comandos rotulados possibilita tanto a utilização de múltiplos comandos, como também reduz a possibilidade de erros de sincronização.

O IMAP pode recuperar apenas os envelopes das mensagens. Essa funcionalidade permite uma melhor utilização do tempo de conexão do cliente, possibilitando, por exemplo, que um usuário exclua uma mensagem da sua caixa de correio sem que seja preciso efetuar a sua transferência para a máquina local, mas apenas baseado nas informações do cabeçalho da mesma.

O IMAP possui também o conceito de cache local independente (*independent local cache*), que é uma abstração, no servidor, da cache existente na máquina do cliente. Esta funcionalidade é responsável pela interatividade entre o servidor e o cliente. Quando o servidor recebe uma mensagem, ele automaticamente envia uma notificação desta para o cliente, que não precisa pedir para verificar se chegaram novas mensagens. Esta funcionalidade de cache local e independente só tem validade quando ambas as máquinas, cliente e servidor, estão conectadas.

O IMAP, mesmo sendo uma solução que visa adicionar muitas funcionalidades importantes ao sistema de acesso e recuperação de mensagens, ainda possui limitações. Ele foi originalmente elaborado para atender a uma classe específica de sistemas, que são os computadores de pequeno porte que estão ligados a uma rede física, ou seja, possuem uma conexão dedicada. Este fato exclui uma grande quantidade de computadores e usuários.

Outro ponto importante a ser ressaltado é que durante o tempo em que o cliente está usando o UA, este mantém uma conexão TCP com o servidor IMAP. Este fato pode ocasionar ônus considerável para o cliente, quando este usa, por exemplo, uma conexão discada.

Uma conexão entre um cliente e um servidor IMAP passa pelos seguintes estados: 1) o estado "não autenticado", quando o cliente ainda não efetuou a sua identificação; 2) o estado "autenticado", após a devida identificação; e 3) o estado "selecionado", quando o cliente selecionar uma pasta de mensagens. Estes estados estão descritos na Figura 2.3 a seguir. Cada estado de uma conexão com o protocolo IMAP possui um subconjunto de comandos válidos.

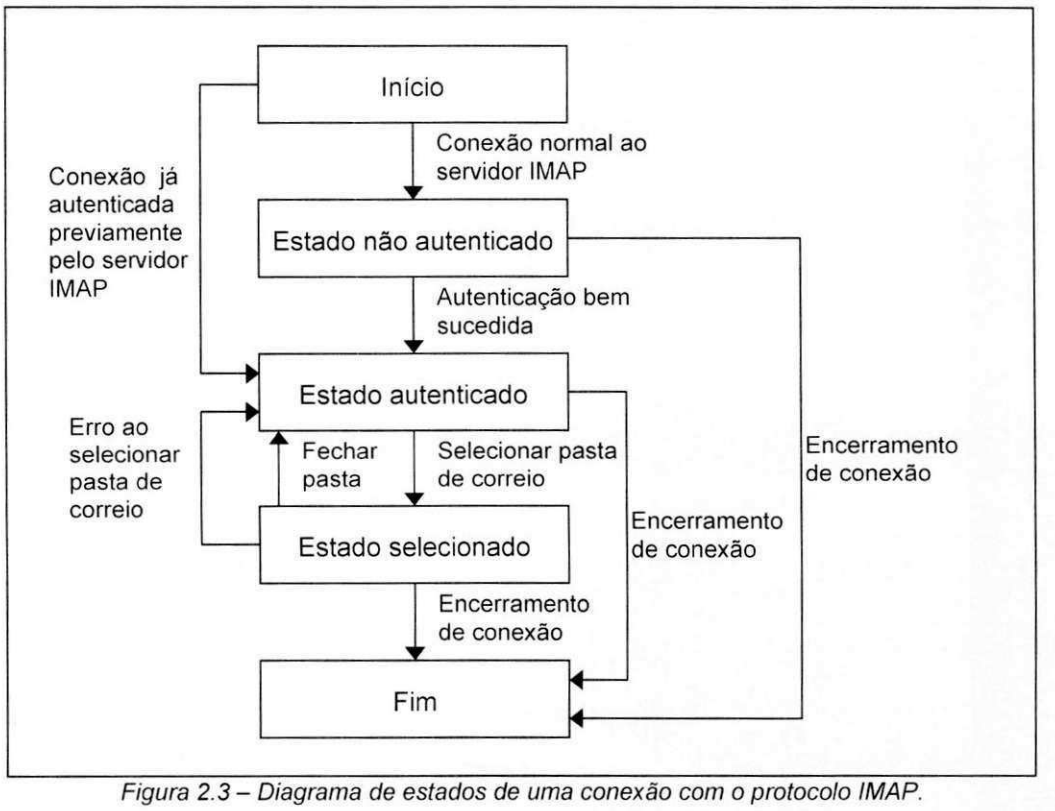

Figura 2.3 – Diagrama de estados de uma conexão com o protocolo IMAP.

Os comandos do IMAP que podem ser utilizados em cada estado da conexão, bem como um breve descrição de cada comando pode ser vista na Tabela 2.3 abaixo.

Estado da Conexão	Comando	Breve descrição
Em qualquer estado	CAPABILITY	Este comando recupera uma lista de *capabilities* do servidor
	NOOP	Não realiza nenhuma função em especial, apenas leva o servidor a responder com uma resposta positiva
	LOGOUT	Requisita o encerramento da sessão
Estado não autenticado	AUTHENTICATE	Permite que os usuários autentiquem-se utilizando um dos mecanismos definidos na RFC 1731 (Kerberos_V4, GSSAPI, ou S/Key)
	LOGIN	Permite que o cliente autentique-se utilizando o nome da caixa de correio e a senha de acesso em formato texto
Estado autenticado	SELECT	Este comando seleciona uma pasta de correio, para que mensagens possam ser recuperadas ou adicionadas nela
	EXAMINE	Tem função idêntica ao comando SELECT, mas com a particularidade que a pasta é aberta apenas para leitura
	CREATE	Cria uma nova pasta com um nome específico
	DELETE	Permanentemente remove uma pasta específica
	RENAME	Altera o nome de uma pasta específica
	SUBSCRIBE	Torna acessível ao usuário uma pasta pública, como no caso de um grupo de notícias (*newsgroups*)
	UNSUBSCRIBE	Cancela o acesso a uma pasta pública
	LIST	Retorna os nomes das pastas disponíveis para o usuário
	LSUB	Retorna a lista do conjunto de pastas públicas ativas para o usuário em questão
	STATUS	Retorna o estado de uma pasta específica
	APPEND	Permite ao usuário adicionar uma mensagem no fim de uma pasta de correio específica
Estado selecionado	CHECK	Pedido para que se estabeleça um *checkpoint* (*na pasta de correio correntemente selecionada*)
	CLOSE	Remove todas as mensagens previamente marcadas para serem excluídas na pasta atual e retorna ao estado autenticado
	EXPUNGE	Permanentemente remove todas as mensagens marcadas para serem excluídas
	SEARCH	Efetua uma busca na pasta atualmente selecionada, por mensagens que possuam um critério específico
	FETCH	Recupera uma ou mais mensagens, podendo esta recuperação ser de mensagens completas ou apenas partes destas
	STORE	Altera os dados associados a uma mensagem
	COPY	Copia uma mensagem (ou várias) para o fim de uma pasta específica
	UID	Seguido pelos comandos COPY, FETCH ou STORE com parâmetros normais, que efetuam as suas funções normais, faz com que estes comandos trabalhem com o UID das mensagens ao invés do número de sequência
		Seguido pelo comando SEARCH com parâmetros normais, faz com que as respostas deste comando contenham UIDs em lugar dos números de sequência das mensagens

Tabela 2.3 – Breve descrição dos comandos do IMAP (1ª revisão da versão 4).

O IMAP também possui uma série de extensões que ajustam o seu funcionamento ou visam suprir deficiências na especificação original. O funcionamento do IMAP é independente destas extensões, visto que a sua estrutura interna é bem

definida, e estas extensões funcionam como módulos de funcionalidade que se agregam ao sistema. Entre as extensões temos, por exemplo, a "*IMAP4 access control list (ACL)*" [Myers97-2]] que define um esquema para restrição de acesso a caixas de correio e pastas, utilizando listas de controle de acesso. A "*IMAP4 optimize-1 extension*" [Myers97-3] busca meios de reduzir o tempo e os recursos requeridos para efetuar algumas operações de um servidor IMAP. A "*IMAP4 quota extension*" [Myers97] permite a administração de limites para a utilização de recursos a serem manipulados através de um servidor IMAP, como por exemplo o espaço reservado em um MS para cada usuário, que pode passar a ser limitado.

Por ser um protocolo com um grande número de funcionalidades, o IMAP é um protocolo complexo, e esta veio a ser a sua grande deficiência. Servidores que implementam o protocolo IMAP tornam-se muito complexos, principalmente aqueles que agregam todas as extensões do protocolo, além de requerer bem mais da máquina que o executará quando comparada a um servidor semelhante baseado no POP.

2.4. Lotus Notes (um exemplo de solução proprietária)

Nesta seção destacaremos um exemplo de uma solução proprietária bastante utilizada, ela é o Lotus Notes [Notes93]. O Notes possui o seu próprio sistema de correio eletrônico, oferecendo aos usuários um meio próprio para o acesso e a recuperação de suas mensagens. O Notes executa em algumas plataformas voltadas para computadores pessoais, como Windows® e OS/2. Podemos definir o Lotus Notes como um gerenciador de informações para grupos de trabalho, com dois principais objetivos: auxiliar no compartilhamento de informações e facilitar a comunicação. A versão do mesmo que foi pesquisada, possui o tipo de licença denominado por *Note Express*.

O Notes gerencia as informações, efetuando basicamente três tarefas: 1) coleta as informações de diversas fontes, 2) organiza as informações para facilitar a sua localização e 3) compartilha estas informações e mantém todos os usuários atualizados quanto as alterações efetuadas.

Uma das grandes ferramentas que o Notes possui é uma base de dados denomina de "Mail". Na visão de um usuário particular a base de dados "Mail" é a sua própria caixa postal, onde estão localizadas as mensagens endereçadas ao mesmo. Esta base de dados especial representa o sistema de correio eletrônico do Notes. Neste sistema, os usuários podem enviar e receber mensagens, além de transportar arquivos. Como no Notes o correio eletrônico é encarado como uma base de dados,

muitas das funções para operação com bases de dados também são utilizadas para o correio eletrônico.

O Notes também possui algumas funcionalidades extras, que são integradas ao serviço de correio eletrônico, entre elas podemos citar: função de "lista de endereços" que armazena o catálogo dos endereços conhecidos para um dado usuário. A função *delivery report*, retorna ao usuário que enviou uma mensagem, algumas informações sobre a entrega da mesma. A função *receipt report*, retorna ao usuário que enviou uma mensagem um aviso indicando que esta foi aberta pelo seu destinatário. O Notes também permite a categorização de documentos como forma de separar determinadas mensagens em classes. Podendo também organizar os documentos na forma de visões diferentes de uma mesma base de dados. O Notes possui também níveis de acesso para garantir a segurança das informações armazenadas em suas bases de dados.

Dentre as funcionalidades do Notes aplicadas a qualquer base de dados que podem ser aproveitadas pelo sistema de correio, destacamos duas que julgamos importantes. A primeira funcionalidade é o que chamamos de visão de uma base de dados, que possibilita a apresentação das informações de uma base de dados de várias formas e com detalhes diferenciados. A segunda funcionalidade é a pesquisa em bases de dados. A pesquisa em uma base de dados é realizada através do comando FIND. Este comando seleciona todos os documentos onde for encontrada a sequência de caracteres que foi informada para a pesquisa. Esta pesquisa textual ainda pode sofrer alguma parametrização, como: *case sensitive* (diferenciando maiúsculas de minúsculas), *accent sensitive* (diferenciando entre caracteres acentuados e não acentuados), *whole word* (palavra inteira), *search within view* (pesquisar no resultado de uma visão), *search selected documents* (pesquisar entre documentos selecionados).

Embora possuindo uma das características desejadas para o serviço de acesso e recuperação de mensagens, que é o acesso de forma distribuída as mensagens de um determinado usuário. O maior problema do Notes é a sua forma de pesquisa, que leva em consideração todo o texto da mensagem, incluindo cabeçalho. O fato não levar em consideração campos específicos das mensagem, ocasiona um esforço maior de pesquisa, enquanto poderíamos ter um acesso direto a informação do campo específico e verificar se possui o critério desejado. Outro desvantagem que tem de ser levado em consideração é que ao optar para utilizar o Notes, o usuário não pode mais escolher qual cliente de correio eletrônico deseja utilizar, pois está preso ao programa cliente do Notes.

2.5. Alguns Problemas na Estrutura do Serviço de Correio Eletrônico Internet

Muitas são as limitações da atual estrutura de correio eletrônico. As limitações sempre foram contornadas, e não se transformaram em problemas mais sérios. Mas, temos acompanhado uma mudança na forma de trabalho com mensagens eletrônicas, o que vem ocasionando o aumento no volume de mensagens que circulam na Internet de uma forma geral. A nossa preocupação é com o usuário que está acompanhando esta taxa crescente de informações circulando na Internet.

O primeiro problema que identificamos é a ineficiência na recuperação das informações de uma caixa de correio. Não há um mecanismo que possibilite ao usuário ter acesso otimizado a tais informações, principalmente se este usuário possuir uma média elevada de mensagens que chegam diariamente a sua caixa postal. A forma mais eficiente, possibilitada pelo POP para o acesso a mensagens, é o de se classificar as mensagens manualmente, identificando-se através de seus cabeçalhos, principalmente a partir do endereço de origem e do assunto da mensagem. A seleção das mensagens neste caso se dá de forma manual e individual, identificando-se quais as mensagens que são relevantes e quais as que não são. Essa tarefa de busca seqüencial e manual de informações é muito desgastante e consome um tempo considerável. Por outro lado, o IMAP através do seu comando SEARCH, embora possibilite um grau de seletividade na recuperação de mensagens, não oferece meios para possibilitar a persistência e consistência destes resultados.

O segundo problema identificado está relacionado com a gerência de caixas postais dos usuários. Mais uma vez, não existe nenhuma funcionalidade nos atuais protocolos de acesso a mensagens que possibilite a gerência automática de caixas de correio dos usuários. O gerenciamento de uma caixa de correio eletrônico pode ser entendido por uma série de procedimentos que visam a organização das mensagens. Podemos organizar as mensagens de um usuário de várias formas, por exemplo: arquivar as mensagens em local específico de acordo com o seu assunto, remover as mensagens que não são mais interessantes ou são velhas, remover as mensagens duplicadas, separar as mensagens urgentes para serem lidas primeiro, etc. A gerência das caixas de correio fica sendo uma função do próprio usuário, a ser realizada manualmente. Estas atividades, como por exemplo o arquivamento de mensagens em pastas e a eliminação de mensagens duplicadas, consomem muito tempo e poderiam ser realizadas automaticamente.

Um terceiro problema está relacionado com a gerência da caixa de entrada de mensagens de cada usuário. Essa gerência se torna um problema quando esta caixa

de entrada de mensagens chega ao seu limite de capacidade. Uma vez estando neste estado, qualquer nova mensagem que chegar destinada a tal caixa de entrada é descartada pelo servidor. Este problema pode ocasionar a perda de informações valiosas para o usuário.

Em vista destas constatações, nos motivamos a desenvolver um novo sistema para o acesso e recuperação de mensagens de correio eletrônico, que seja eficiente, ao mesmo tempo que seja uma solução de relativa simplicidade de implementação e utilização. Essa solução deve possibilitar aos usuários do sistema de correio eletrônico o acesso seletivo a informações armazenadas na forma de mensagens de correio eletrônico sem necessitar de recorrer a soluções complexas.

Capítulo 3

Uma Alternativa para o Acesso e Recuperação de Mensagens de Correio Eletrônico

Em nossos estudos identificamos várias classes de usuários cujas necessidades não são satisfatoriamente atendidas pelos protocolos de acesso e recuperação de mensagens existentes. Em vista destas necessidades identificadas, estamos propondo um novo protocolo.

Para dar início à especificação deste novo protocolo para o acesso e recuperação de mensagens, o *Smart Post Office Protocol*, ou simplesmente SPOP, foi identificado o público alvo para o mesmo. Mesmo sendo uma solução genérica, este protocolo visa satisfazer algumas necessidades específicas, as quais julgamos muito importantes em termos de acesso eficiente às mensagens de correio eletrônico. Para apresentar algumas características do público alvo do protocolo SPOP, o dividimos em classes de usuários.

A primeira classe é formada por usuários que trabalham com um grande volume de informação armazenada na forma de correio eletrônico, mas com tempo limitado para efetuar pesquisas sobre essa informação. Nesta primeira classe encontram-se, por exemplo, profissionais com algum poder de decisão nas suas organizações e que necessitam do maior número de informações para respaldar as suas decisões. A principal característica desta classe de usuários está no fato de seus componentes possuírem grandes limitações no tempo alocado para a realização de suas tarefas, por isso necessitam de agilidade no acesso e recuperação de mensagens de correio eletrônico.

A segunda classe é constituída por usuários que utilizam o correio eletrônico como o seu principal meio de comunicação. Nesta classe se encontram, por exemplo, empresas que trabalham com o comércio eletrônico através da Internet, e que utilizam o correio eletrônico para a comunicação com os seus clientes. As características que destacamos desta classe são duas: a primeira é o valor nas informações contidas nas mensagens eletrônicas, e a segunda é o volume destas mensagens.

A terceira classe é formada por usuários que desejam acessar suas mensagens eletrônicas de vários pontos da rede. Fazem parte desta classe todos os usuários que não possuem uma máquina individual para acessar e manipular as suas mensagens, e por isso, não podem estar presos a uma estrutura que utilize uma UA específico.

A quarta classe que especificamos é a dos usuários individuais que não conseguem administrar manualmente sua caixa de correio e suas pastas de arquivamento, em virtude do grande volume de mensagens. Nesta classe encontram-se, por exemplo, os usuários que se subscrevem em várias listas de discussão pertinentes as suas funções. O maior problema desta classe de usuários está na administração da sua caixa de correio, e a decorrente superlotação da mesma.

Na quinta e última classe que especificamos encontram-se os usuários que necessitam de mais funcionalidades do que as oferecidas pelo POP, e não estão dispostos a pagar o preço de ter uma solução mais complexa e que possui mais exigências, como o IMAP. O fato é que o servidor POP necessita de menos recursos da máquina que um servidor IMAP. Nesta classe encontram-se, por exemplo, usuários de provedores de acesso Internet que estariam interessados em funcionalidades que os auxiliassem a trabalhar mais eficientemente com suas mensagens de correio eletrônico, e que possuem do outro lado os provedores de acesso relutando em oferecer um serviço que iria requerer mais espaço de armazenamento, bem como maior poder de processamento nas máquinas que viessem a prover este serviço.

Os principais requisitos para o protocolo SPOP são voltados para atender às necessidades específicas destas cinco classes de usuários do sistema de correio eletrônico. O protocolo SPOP visa oferecer meios para que estas classes de usuários possam otimizar o seu acesso e recuperação a mensagens de correio eletrônico.

3.1. Requisitos do SPOP

O SPOP tem quatro requisitos principais. O primeiro é ser uma solução que ofereça a possibilidade de uma recuperação de mensagens seletiva e eficiente. Ou seja, uma forma de recuperação em que o usuário possa obter rapidamente e com poucas interações a informação de seu interesse no momento do acesso ao sistema de correio eletrônico. Com uma recuperação seletiva e eficiente, os usuários da primeira classe, que possuem pouco tempo para trabalhar com um grande número de informação, são beneficiados, pois podem recuperar eficientemente a informação já previamente selecionada segundo algumas das suas necessidades particulares (as que já são conhecidas). Da mesma forma, os usuários da quarta classe, que possuem

um grande volume de informação, são beneficiados por poderem contar com a seleção das suas mensagens, podendo de certa forma priorizar a leitura das mensagens que julgar importantes. Ainda são beneficiados os usuários da segunda classe, que têm no correio eletrônico o seu principal meio de comunicação, pois ganha a facilidade de selecionar e recuperar as mensagens de acordo com o seu remetente e assim coordenar melhor as novas mensagens, bem como as respostas.

O segundo requisito é ser uma solução que ofereça a facilidade de organização das mensagens. Uma forma de auxiliar o usuário na administração da sua caixa de correio, programando com antecedência a classificação das suas mensagens em pastas. Esta classificação deverá acontecer de forma automática, sem requerer que o usuário informe sempre que deseja que suas mensagens sejam classificadas, mas apenas uma única vez e especificando como isso deve acontecer. Com a classificação programada das mensagens são beneficiados os usuários da primeira classe, pois saberão onde poderão encontrar as mensagens que desejam a cada instante. São beneficiados principalmente os usuários da segunda classe, pois como o correio eletrônico é o seu meio de comunicação mais importante, é importante que tenha essas mensagens devidamente organizadas, para que possa dispor dessas informações eficientemente. Da mesma forma, são beneficiados os usuários da quarta classe, os quais possuem um grande volume de informação, que podem organizar as suas mensagens de modo que possam encontrar o que precisam.

O terceiro requisito é ser uma solução de relativa simplicidade quanto ao seu uso e implementação. Um protocolo com poucos comandos e comandos direcionados a resolver os problema aos quais se propõe atender. Um protocolo com uma interface de comandos simples, possibilitando a construção de clientes e servidores com relativa simplicidade. O fato de se ter uma solução simples quanto ao seu uso e implementação, redunda como benefício aos usuários da quinta classe, os quais possuem poucos recursos computacionais e não estariam dispostos a sobrecarregar o seu sistema com uma solução que possua um grau maior de complexidade.

O quarto e último requisito principal para o protocolo SPOP é ser uma solução que se adapte à atual estrutura de correio eletrônico. Um protocolo que se adeque perfeitamente à estrutura nativa do correio eletrônico, não obrigando o usuário a efetuar nenhuma alteração na sua estrutura de correio, bastando inserir o servidor e o cliente SPOP e passar a usufruir das suas funcionalidades. Sendo uma solução adaptável a atual estrutura de correio eletrônico, o SPOP beneficia os usuários da terceira classe, pois oferece uma estrutura de acesso distribuído sem que seja necessária a modificação da estrutura de armazenamento das mensagens. Da mesma

forma que beneficia os usuários da quinta classe, pois o SPOP necessitará de poucos recursos a mais que o POP, que é a solução mais simples para o acesso e recuperação de mensagens de correio eletrônico Internet.

3.2. Especificação do SPOP

Nesta seção descreveremos algumas direções tomadas para atender aos requisitos anteriormente levantados para o protocolo SPOP. Apresentaremos inicialmente um novo conceito desenvolvido para dar suporte às funcionalidades do protocolo SPOP. Além disso, definiremos a interface de comandos do SPOP, fazendo uma descrição de cada comando, seus argumentos, restrições, sintaxe, respostas e exemplos de utilização.

3.2.1. Pastas de Classificação Programada

Para atender aos requisitos de recuperação seletiva e classificação programada desenvolvemos o conceito de "pastas com classificação programada", que são pastas de arquivamento que vão possibilitar a classificação programada de mensagens de acordo com as suas informações. Denominamos esta pastas especiais de *Smart Folders*, ou simplesmente *Sfolders*. Um *Sfolder* é uma pasta de mensagens com uma regra associada. Esta regra especifica uma série de critérios que as mensagens deverão ter que satisfazer para serem automaticamente movidas para esta pasta. Esta busca por critérios específicos acontece no lado do servidor; o cliente apenas utiliza os resultados das buscas na forma de uma pasta especial denominada *SFolder*.

Cabe nesse ponto fazer um aparte para explicar o motivo da decisão de se optar por realizar a busca de mensagens no servidor e não no cliente. Uma primeira razão está no fato que a busca de mensagens no servidor propicia que os clientes para este protocolo sejam mais simples, visto que não precisam incluir nos seus códigos mecanismos de busca. Uma segunda razão está no fato de ser gerado um grande tráfego na rede quando realizamos a pesquisa no cliente, ao mesmo tempo que temos o acesso a partir de diversas localidades, pois cada cliente que acessa a caixa de correio e deseja realizar uma busca deve possuir uma cópia atualizada desta (ocorre que toda a caixa de correio é transmitida várias vezes através da rede).

Definimos uma linguagem especial para a construção de regras para a especificação de *Sfolders*, esta linguagem chama-se *Smart Folder Definition Language*, ou simplesmente SFDL, a qual descreveremos com mais detalhes no decorrer deste documento.

A utilização de um *Sfolder* se dá da seguinte maneira. Em primeiro lugar um cliente SPOP utiliza um comando do servidor para a sua criação, onde um dos argumentos deste comando é a regra de formação, especificando os critérios de classificação. Um exemplo de um conjunto de critérios de classificação de mensagens seria a especificação do subconjunto das mensagens que tiveram como remetente o usuário "fellipe" do domínio "dsc.ufpb.br", que contêm o texto "dissertação" no seu assunto, que forem antes da data "20/06/1999". Os critérios definidos acima seriam úteis para um professor orientador que está esperando do seu aluno Fellipe a submissão de sua dissertação, cujo prazo de entrega encerra no dia 10 de maio de 1999.

Uma vez enviado o comando de criação do *Sfolder,* o servidor SPOP realiza uma busca nas mensagens que satisfaçam a estes critérios, e automaticamente move para a pasta em questão as mensagens encontradas nesta pesquisa. Na prática o *Sfolder* é uma pasta lógica, pois as mensagens continuam sendo armazenadas no MS da mesma forma. O servidor SPOP é o responsável por guardar as informações das pastas do usuário, suas regras de formação, e as mensagens contidas nestas pastas. A cada nova mensagem recebida, o servidor SPOP investiga se esta deve estar em alguma pasta específica. Devido aos *Sfolders* serem uma estrutura lógica, as mensagens podem fazer parte de mais de um *Sfolder* distinto (sem a necessidade de duplicação de mensagens), desde que satisfaçam às regras associadas a esses *Sfolders*. Para tornar o processo com *SFolders* mais eficiente, do ponto de vista do servidor, idealizamos uma estrutura auxiliar que guarda a informação sobre quais *SFolders* uma determinada mensagem está vinculada; esta estrutura auxilia o servidor a atualizar as informações dos *SFolder* quando uma mensagem em particular for removida. Uma vez criado um *Sfolder*, o usuário pode a qualquer momento recuperar as mensagens com os critérios especificados na criação do *Sfolder*, com a garantia da consistência da recuperação com relação às mensagens que foram recebidas depois da criação do mesmo.

Um *SFolder* pode conter um atributo especial, atributo este que dará ao *SFolder* um conjunto de características específicas. Os atributos especiais podem ser dois, o primeiro deles é INBOX, o qual faz com que o *SFolder* se comporte como a caixa de entrada padrão do sistema de correio eletrônico. As características da INBOX são as seguintes: não possui critérios específicos de classificação e todas as mensagens que não estiverem vinculadas a outros *SFolders* permanecem vinculadas a INBOX. O outro atributo especial é TRASH; este atributo faz com que todas as

mensagens vinculadas a este *SFolder* sejam excluídas quando for encerrado o trabalho com a referida caixa postal eletrônica.

Para ser adaptável à atual estrutura de correio eletrônico, o SPOP utiliza as definições do SMTP (RFC 821) quanto a estrutura do MS e não impõe nenhuma padronização quanto à forma de armazenamento das mensagens. Por ser adaptável, o SPOP pode ser instalado em servidores de correio eletrônico que já possuam outras formas de acesso e recuperação de mensagens, a saber o POP e o IMAP. O SPOP também oferece suporte às pastas de arquivamento de mensagens do IMAP. No IMAP além das pastas comuns e sua estrutura hierárquica associada, existem também as pastas públicas, que são estruturas para possibilitar, por exemplo, a manipulação de grupos de notícias (NEWS) [Kanto86]. Essa característica faz com que usuários dos outros sistemas passem a utilizar o SPOP de forma transparente e sem perda quanto à qualidade do serviço anteriormente usufruído.

A estrutura dos *Sfolders* torna o SPOP uma solução ágil, pois com a criação destas pastas especiais com regras associadas, temos as mensagens da nossa caixa de correio classificadas de acordo com os critérios mais importantes para o usuário em questão. Esta classificação das mensagens faz com que o referido usuário saiba onde ir buscar as mensagens que deseja a cada momento, recuperando-as com agilidade. O poder de classificação de mensagens obtido com os *Sfolders* possibilita que as mensagens mais importantes (de acordo com critérios pessoais) tenham prioridade de leitura. Pois, o usuário pode criar um *Sfolder* especificando as características das mensagens que são mais importantes para o mesmo e configurar o seu sistema de correio eletrônico para abrir primeiramente este *Sfolder*. A classificação de mensagens também auxilia o usuário quando o volume de suas mensagens for grande, se este usuário não tiver tempo de ler todas as suas mensagens. Neste caso, o usuário pode priorizar o que deseja ler pois pode recuperar os *Sfolders* contendo as mensagens que lhe interessarem no momento.

A proposta dos *Sfolders* é que estes sejam armazenados no servidor de correio eletrônico, onde também serão armazenadas as mensagens de correio eletrônico dos usuários do sistema. Esta centralização no armazenamento possibilitará que os usuários acessem de forma distribuída as suas mensagens, tendo acesso aos mesmos *Sfolders* a partir de qualquer cliente de correio eletrônico que suporte o protocolo SPOP. Com todas essas características o SPOP configura-se como uma solução intermediária entre o POP e o IMAP, trazendo a simplicidade que é um dos pontos fortes do POP e a flexibilidade e distribuição do acesso às mensagens, que é um dos pontos fortes do IMAP.

Para prover a simplicidade objetivada pelo SPOP foi definida um interface com poucos comandos, mais intuitivos e com sintaxe e respostas mais diretas. Comprovaremos este fato na próxima seção em que descreveremos todos os comandos do SPOP.

3.2.2. Comandos do SPOP

A comunicação entre cliente e servidor através do protocolo SPOP será orientada a linhas em ASCII. Nas exemplificações dos comandos do SPOP utilizaremos a seguinte convenção: as linhas que indicarem um comando enviado pelo cliente, iniciarão com os caracteres "C:", e semelhantemente, as linhas que indicarem uma resposta enviada pelo servidor iniciarão com os caracteres "S:".

Como nos outros protocolos para acesso e recuperação de mensagens, a sessão entre o servidor e o cliente é dividida em estágios. Cada estágio de conexão tem associado a si um subconjunto próprio de comandos válidos. O primeiro estágio chama-se "NÃO AUTENTICADO". Neste estágio o servidor deu por iniciada a sessão mas não existe nenhum usuário autenticado. O segundo estágio chama-se "AUTENTICADO". Uma sessão chega a esse estágio após o cliente que iniciou a sessão autenticar-se como um usuário do sistema. Neste estágio o cliente já tem acesso às mensagens da sua pasta de entrada. Salientamos aqui a diferença entre pasta de correio e pasta de entrada, pois esta diferenciação será importante no decorrer do documento.

No POP, como não existe o conceito de pastas especiais para arquivamento de mensagens, as mensagens ao chegarem à caixa postal eletrônica do usuário (MAILBOX) são depositadas na (INBOX) e permanecem nesta até serem excluídas; neste caso a MAILBOX e a INBOX simbolizam o mesmo local. Já no IMAP, existe o conceito de pastas de arquivamento. Após uma mensagem chegar ao MAILBOX de um usuário e ser depositada na INBOX, o IMAP possibilita ao usuário mover esta mensagem para uma outra pasta. Neste caso o termo MAILBOX serve para referenciar o conjunto das pastas que compõem a caixa postal eletrônica de um usuário, incluindo a INBOX. Como o SPOP também possui o conceito de pastas de arquivamento, após as mensagens serem recebidas e depositadas na INBOX, o SPOP possibilita que estas mensagens sejam vinculadas automaticamente a uma outra pasta diferente da INBOX. Portanto, quando utilizarmos o termo "pasta de correio" estaremos nos referindo à conta do usuário no sistema de correio (*MAILBOX*) e o conjunto das pastas associadas a esta, e quando utilizarmos o termo pasta de

entrada estaremos nos referindo à pasta padrão do sistema de correio de um referido usuário (*INBOX*).

O terceiro estágio chama-se "SELECIONADO". A sessão passa para este estágio depois de o cliente já autenticado haver selecionado uma pasta específica para ser explorada. O quarto e último estágio chama-se "ENCERRANDO". Uma sessão entra neste último estágio quando o cliente faz o pedido ao servidor para que sua sessão seja encerrada. O servidor utiliza o estágio final para fazer as últimas alterações na caixa postal do cliente. A Figura 3.1, logo abaixo, nos mostra uma representação gráfica dos estágios possíveis de uma conexão através do protocolo SPOP e como ela pode ser atingida. O único estágio no qual não são aceitos comandos é o estágio "Encerrando". Neste estágio o servidor atualiza as informações sobre as mensagens e os *SFolders* do usuário, as mensagens marcadas para serem excluídas são excluídas fisicamente e os *SFolders* que continham esta mensagem são atualizados.

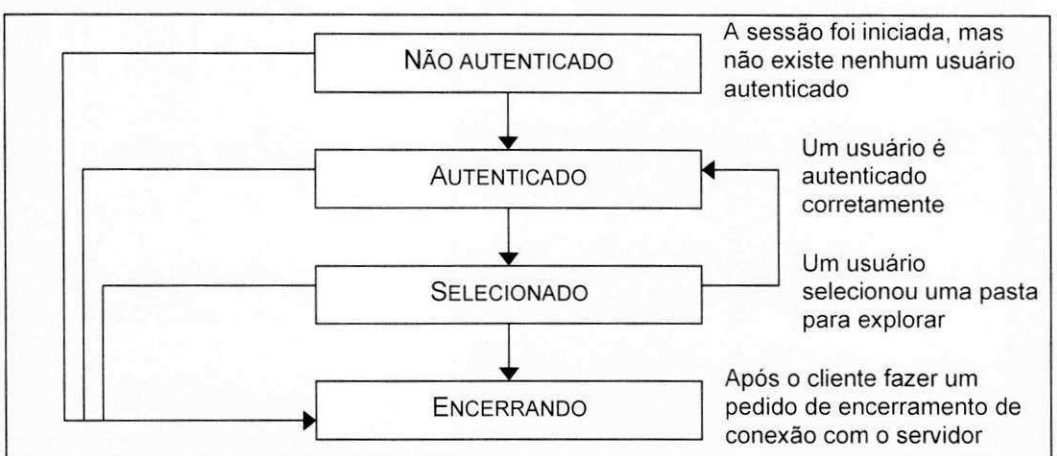

Figura 3.1 – Diagrama de estágios de uma conexão com o protocolo SPOP.

As respostas do servidor SPOP iniciam com um sinal de positivo ("+") ou negativo ("-"). Esta sinalização tem dois objetivos básicos fundamentais. Primeiro, representam o caráter da resposta que está sendo recebida, ou seja "positiva" ou "negativa". Esta indicação do caráter da resposta ajudará a um cliente, que implementa o protocolo SPOP, a interpretar a resposta do servidor SPOP. Em segundo lugar, elas servem para marcar o início de uma nova resposta do servidor, visto que as respostas do servidor SPOP podem ser compostas de várias linhas. Esta indicação de início de respostas do servidor, pode ser útil ao cliente SPOP que deseje ignorar o contexto de uma resposta de várias linhas, pois o cliente pode descartar todas as informações até receber um dos dois sinais representando o início de uma nova resposta.

Abaixo, na Tabela 3.1, temos uma breve descrição de todos os comandos do protocolo SPOP. Tais comandos serão discutidos na sequência desta seção.

Estágio da Conexão	Comando	Breve descrição
Em Qualquer Estágio	END	Pode ser utilizado a qualquer momento da conexão e representa o pedido para o encerramento da conexão
	CONFIRM	O cliente usa este comando para confirmar o encerramento de uma conexão, mesmo quando ainda existem tarefas pendentes do lado do servidor
	PING	Comando que requer uma resposta do servidor. É utilizado para testar a comunicação entre o cliente e o servidor
Estágio **Não Autenticado**	MAILBOX	Permite a forma mais simples de autenticação de usuários para o acesso a uma caixa de correio (modo texto)
	PASS	É utilizado para completar o comando MAILBOX. É através deste que o usuário informa a sua senha de acesso
	AUTHENTICATE	Este comando permite ao usuário escolher um método mais seguro de autenticação implementado internamente pelo SPOP, como por exemplo o Kerberos
	AUTHSERVER	Permite que o servidor SPOP faça uso de servidores de autenticação de usuários independentes
Estágio **Autenticado**	STATUS	Retorna informações gerais sobre a caixa de correio
	CREATE	Utilizado para criar um novo *SFolder* (pasta especial)
	RENAME	Altera o nome de um *SFolder*
	SHOW	Exibe a lista das subpastas e mensagens contidas na pasta atualmente em uso
	FETCH	É utilizado para recuperar mensagens específicas
	EXAMINE	Semelhantemente ao FETCH é utilizado para recuperar mensagens. Neste caso é recuperado um conjunto de mensagens vinculadas a um *SFolder*
	HEADER	Recupera um determinado número de linhas de uma mensagem específica
	REMOVE	Utilizado para marcar as mensagens que serão excluídas ao fim da seção, também pode ser utilizado para remover pastas especiais (incluindo *SFolders*)
	DELFOLDER	Remove uma pasta de arquivamento
	UNMARK	Restaura uma série de mensagens marcadas pelo comando REMOVE; só é válido para mensagens marcadas para serem removidas na sessão corrente
	UIDM	É utilizado para recuperar os identificadores únicos para uma série de mensagens
	TRIGGER	É um comando que cria e remove condições de disparo para determinadas ações predefinidas
	SELECT	Seleciona uma determinada pasta para ser explorada
	DESELECT	Volta um nível na hierarquia das pastas de arquivamento
Estágio **Selecionado**	STATUS, CREATE, RENAME, SHOW, FETCH, EXAMINE, HEADER, REMOVE, DELFOLDER, UNMARK, UIDM, WHEN, SELECT e DESELECT	Como já vistas anteriormente, com uma diferença apenas: no estágio autenticado a pasta em questão é a própria caixa de entrada do usuário; já no estágio selecionado a pasta em questão foi anteriormente selecionada pelo comando SELECT
Estágio **Encerrando**	Não são permitidos comandos neste estágio da conexão	

Tabela 3.1 – Breve descrição dos comandos do SPOP.

A sessão com o servidor inicia-se com a abertura de uma conexão TCP utilizando-se a porta TCP 1110. Ao confirmar a abertura da conexão o servidor SPOP retorna uma mensagem de saudação.

Exemplo:

```
S: +SPOP READY
```

A mensagem de saudação indica que o servidor está em pleno funcionamento e está pronto para receber os próximos comandos do cliente.

Quando o cliente faz o pedido para que a conexão seja encerrada, ele recebe uma indicação se o servidor SPOP já terminou ou não as atividades relativas ao referido cliente. Logo após o servidor envia uma mensagem informando que está pronto para encerrar a conexão.

Exemplo:

```
S: +SPOP SAYS BYE, BYE
```

Em resposta aos comandos do cliente o servidor SPOP pode enviar basicamente três tipos de respostas. O primeiro tipo é quando o comando que o cliente enviou foi completamente satisfeito. Esta resposta inicia-se com o servidor respondendo positivamente ao cliente, informando que tudo está correndo bem.

Exemplo:

```
S: +SPOP OK
```

O segundo tipo de resposta refere-se a uma das seguintes situações: o comando enviado pelo usuário não é permitido para ser utilizado no atual estágio da sessão; o comando simplesmente não existe; ou os argumentos submetidos são ilegais.

Exemplo:

```
S: -SPOP ERR
```

O terceiro tipo de resposta refere-se a comandos válidos utilizados pelo usuário para o atual estágio da sessão, mas que não obtiveram sucesso em sua execução.

Exemplo:

```
S: -SPOP FAILED
```

A diferença entre os dois tipos de respostas negativas do servidor SPOP, é que a resposta que informa um erro, está apontando que o comando referido não pode, por algum motivo especial, iniciar a sua execução. Já uma resposta que informa uma

falha, indica que o comando requisitado falhou durante a sua execução. São inúmeros os motivos que podem causar erros e falhas no sistema de transferência de mensagens. As repostas do servidor SPOP visam ser claras e objetivas, levando ao cliente a uma perfeita noção dos possíveis erros que podem advir da utilização dos seus comandos. As repostas indicando falhas no servidor serão utilizadas em trabalhos futuros visando agregar características como tolerância a faltas, alta confiabilidade de funcionamento e recuperação de falhas.

Além dessas respostas básicas o servidor pode retornar detalhes sobre a operação que indicarão mais precisamente o que ocorreu, e no casos de falhas e erros quais devem ser as providências que deverão ser tomadas.

No item referente à sintaxe dos comandos usaremos a seguinte notação: os colchetes ("[" e "]") delimitarão argumentos opcionais e os sinais de "menor que" e "maior que" ("<" e ">") delimitarão os argumentos obrigatórios. Os argumentos serão formados por cadeias de caracteres ASCII. Os argumentos representativos são apenas breves explicações sobre os reais argumentos que deverão acompanhar o comando em questão.

Outra notação utilizada na sintaxe dos comandos do servidor SPOP é o operador de condição "ou", neste caso representado pela barra vertical ("|"). Quando os comandos do servidor SPOP possuem mais de uma forma de sintaxe, as opções de estilo para um comando aparecem separadas pelo operador lógico.

O servidor oferece ao cliente SPOP a opção de aumentar a velocidade de comunicação com este. Opcionalmente o servidor aceita que múltiplos comandos sejam recebidos, sem que se faça necessária a espera de uma resposta de um referido comando para que se possa enviar o próximo. Para possibilitar esta funcionalidade o servidor SPOP aceita que os comandos enviados ao servidor sejam rotulados, semelhantemente ao IMAP. A facilidade de se utilizar rótulos nos comandos enviados ao servidor SPOP é opcional, mas é muito útil para algumas funcionalidades que o cliente pode desenvolver como a ressincronização da comunicação a qualquer momento e identificação de respostas perdidas.

Os rótulos aceitos pelo servidor SPOP possuem a seguinte forma: "AA001". São duas letras e três números, que não podem ser repetidos durante uma conexão, exceto quando estourar o número máximo de rótulos possíveis (mais de 600.000 rótulos). A escolha deste formato de rótulos foi baseada no critério de conseguir o maior conjunto de rótulos possíveis, com o melhor custo benefício. Estes rótulos são

sempre crescentes, seguindo uma ordenação lexicográfica, não necessariamente seqüenciais. Por exemplo: "AA000 < AA001 < ... < AA999 < AB000 < ... < ZZ999".

Exemplo:

```
C: AA000 <comando 1>
C: AA001 <comando 2>
C: AA005 <comando 3>
S: +AA000 <resposta ao comando 1>
S: +AA005 <resposta ao comando 3>
S: +AA001 <resposta ao comando 2>
```

Nos exemplos que utilizaremos para demonstrar o funcionamento dos comandos do servidor SPOP não utilizaremos rótulos, para que o perfeito entendimento dos comandos do servidor não seja prejudicado.

3.2.2.1. Comandos permitidos no estágio "NÃO AUTENTICADO"

Este é o primeiro estágio da sessão estabelecida entre um cliente e um servidor SPOP, iniciado após a conexão de rede entre estes. Antes de ser autenticado, o cliente tem as opções de encerrar a conexão, autenticar-se para utilizar uma caixa de correio válida no sistema ou fazer uso de comandos de controle de conexão.

I. Comandos para autenticação de usuários

Existem várias formas possíveis para a autenticação de um cliente. O mesmo deve escolher aquela que melhor lhe convier.

a) MAILBOX

Sintaxe:

```
MAILBOX <caixa de correio>
```

Argumentos:

Nome do usuário a quem pertence a caixa de correio.

Restrições:

O comando MAILBOX só pode ser utilizado no estágio "NÃO AUTENTICADO". O nome informado tem que ser de um dos usuários cadastrados no sistema e obedecendo as restrições de formato impostas pelo sistema. O comando MAILBOX possui uma restrição quanto a sua utilização, pois a sua forma de autenticação, em conjunto com o comando PASS oferece problemas de segurança. Nesta forma de autenticação a senha da caixa de correio do usuário trafega na rede no formato texto e

sem criptografia. Este método de autenticação só é indicado para redes que já ofereçam mecanismos de segurança externos ao serviço de correio eletrônico.

Possíveis respostas:

`+SPOP OK, AWAITING PASS` - quando existe a referida caixa de correio.

`-SPOP ERR, NO SUCH MAILBOX` - quando não existir a caixa de correio.

`-SPOP FAILED` - quando o sistema não consegue autenticar um usuário.

Exemplos:

```
C: MAILBOX spop
S: -SPOP ERR, NO SUCH MAILBOX
C: MAILBOX fellipe
S: +SPOP OK, AWAITING PASS
```

Discussão sobre o comando:

Este método de autenticação é o mais simples existente. A autenticação do usuário é feita em parceria com o comando PASS, que é esperado após a utilização deste. Não existe nenhum mecanismo de segurança envolvido nesta transação, portanto só é indicada para redes que já possuam mecanismos de segurança próprios.

Quando a resposta do servidor for positiva, existe então a indicação que tal usuário está cadastrado no sistema de correio e para que seja completada a autenticação deste usuário o comando PASS tem que vir em seqüência. Quando a resposta do servidor for negativa e indicar um erro (ERR), isto representa que tal usuário não está cadastrado no sistema de correio. Já uma resposta negativa indicando uma falha (FAILED), assinala que este tipo de autenticação não é permitida para a referida caixa de correio ou ocorreu algum problema durante a autenticação.

b) PASS

Sintaxe:

`PASS <senha de acesso>`

Argumentos:

A senha de acesso à caixa de correio anteriormente informada com o comando MAILBOX.

Restrições:

Este comando só pode ser utilizado após um comando MAILBOX que obteve resposta positiva. A senha não pode conter caracteres especiais do sistema de correio

eletrônico. A utilização deste comando está restrita ao estágio "NÃO AUTENTICADO" de uma sessão entre um cliente e um servidor SPOP.

Possíveis respostas:

`+SPOP OK, WELCOME` - quando a senha for reconhecida para a caixa de correio.

`-SPOP ERR, WRONG PASS` - quando a senha informada não confere.

`-SPOP FAILED` - quando não for possível realizar a autenticação.

Exemplos:

```
C: MAILBOX fellipe
S: +SPOP OK, AWAITING PASS
C: PASS passlipe
S: -SPOP ERR, WRONG PASS
C: MAILBOX fellipe
S: +SPOP OK, AWAITING PASS
C: PASS senha1
S: +SPOP OK, WELCOME
C: PASS test
S: -SPOP FAILED
```

Discussão sobre o comando:

O comando PASS é utilizado apenas em conjunto com o comando MAILBOX. Os dois em conjunto (MAILBOX e PASS) têm a função de autenticar um usuário para utilizar uma caixa de correio do sistema.

Uma importante ressalva que tem que ser feita é com relação aos espaços em branco utilizados no campo de argumentos. O comando PASS só aceita um argumento, que é a senha do usuário. Se forem digitados espaços em branco, estes serão considerados como parte integrante das senhas de acesso.

c) AUTHENTICATE

Sintaxe:

`AUTHENTICACE <nome do mecanismo de autenticação>`

Argumentos:

O argumento para este comando é o nome do mecanismo de autenticação que deverá ser utilizado. Os mecanismos de autenticação aceitos dependerão da implementação do servidor SPOP. É sugerido que qualquer implementação do SPOP contenha ao menos os seguintes mecanismos para autenticação: BASE64, KERBEROS e MD5 (todos estes baseados em criptografia). Opcionalmente, os nomes

dos mecanismos de autenticação poderão ser seguidos dos caracteres "_V", seguidos de um número que indica a versão do mesmo, como por exemplo: KERBEROS_V4.

Restrições:

Só pode ser executado no estágio "NÃO AUTENTICADO" da sessão. O nome do mecanismo de autenticação tem que ser aceito pelo servidor SPOP.

Possíveis respostas:

`+SPOP OK`

`...<diálogo para a autenticação do usuário>...`

`SPOP DONE!` – quando o servidor SPOP reconhecer o referido mecanismo de autenticação e o processo de autenticação for concluído com sucesso.

`-SPOP ERR` – quando o mecanismo não for reconhecido ou não estiver disponível.

`+SPOP OK`

`...<diálogo entre o cliente e o servidor de autenticação>...`

`SPOP FAILED!` – quando ocorrer um erro durante a autenticação do usuário ou a senha de acesso ou nome do usuário forem informados incorretamente.

`+SPOP BREAK OFF` – quando o processo de autenticação for interrompido.

Exemplos:

```
C: AUTHENTICATE KERBEROS_V4
S: +SPOP OK
S: +AmFYig==
C:
   BYG7bB&t8bYG87tbkjbkBIY(798hH(&98uyhOH(79u;mNkfT$^fjhgb
   vwm
   GY%352648&&^)pk[09=09(78OLJlkghi685r7^TUgjgr75$7654758y
   OJHNouj0980979087698&^%iyiughiut8&%^(70jlkj)(780HGhfH
S: +or//EoAADZI=
C: DffdtJjug765450M090jK987g76Tg==
S: SPOP DONE!
```

Discussão sobre o comando:

O comando AUTHENTICATE permite que um usuário autentique-se de uma forma mais segura que a solução implementada pelos comandos MAILBOX e PASS. O comando AUTHENTICATE possibilita que um cliente escolha um dos métodos de autenticação implementados internamente pelo servidor SPOP. Os nomes utilizados

para referenciar os mecanismos devem ser do conhecimento do servidor e do cliente, pois a sua referência é direta e o servidor não faz aproximações de nomes.

Portanto uma vez o cliente fazendo referência a um determinado mecanismo de autenticação, se este for suportado pelo servidor, o protocolo para a autenticação do usuário com o referido mecanismo é iniciado. A troca de mensagens do protocolo é iniciado com o servidor enviando a continuação da resposta ao pedido do cliente precedido do sinal "+" seguido de uma cadeia de texto codificado em um mecanismo padrão como o BASE64. Isto também ocorre com as respostas do usuário.

O servidor SPOP disponibiliza um artifício para o cliente, para possibilitar que o processo de autenticação seja encerrado a qualquer momento. Para tal o cliente utiliza-se de uma linha de comando com o caracter asterisco ("*") sem codificação. Uma vez o servidor recebendo um sinal para a interrupção do processo, ele envia uma confirmação e retorna ao modo de estar apto a receber outros comandos do cliente.

O comando AUTHENTICATE tem a mesma forma que o comando de mesmo nome do IMAP, e como tal foi incorporado ao SPOP para prover mecanismos de autenticação mais confiáveis. Com este comando, o cliente pode estabelecer uma ordem decrescente em termos de grau de confiabilidade de mecanismos de autenticação. Ou seja, inicialmente o cliente poderia tentar em primeira instância a execução de um mecanismo mais seguro. Uma vez o servidor SPOP não provendo este tipo determinado de autenticação, o cliente poderia tentar um mecanismo de um grau de confiança menor, e por fim, não existindo outra opção, realizaria a autenticação com os comandos MAILBOX e PASS.

d) AUTHSERVER

Sintaxe:

```
AUTHSERVER <servidor de autenticação> [<máquina>:<porta>]
```

Argumentos:

É importante ressaltar que este comando tem a mesma função que o comando AUTHENTICATE, com a diferença básica que neste comando a implementação do mecanismo de autenticação é externo ao servidor SPOP e de responsabilidade de um servidor próprio para tal fim. Através deste comando é passado o nome do servidor que será responsável pela autenticação dos usuários do sistema. Serão informados também o endereço da máquina onde se encontra o servidor (seja o nome da máquina ou o seu número IP) e a porta na qual o servidor está aguardando conexões.

Restrições:

Só pode ser executado no estágio "NÃO AUTENTICADO" da sessão. Para configurar o servidor SPOP para passar a aceitar um certo servidor de autenticação, situado em uma determinada máquina, deve ser editado o arquivo de configurações do servidor (denominado de spop.conf, ou simplesmente spop.cnf). No arquivo de configurações do servidor deverão constar os nomes dos servidores aceitos pelo servidor SPOP para efetuar a autenticação de seus usuários, bem como a máquina em que estes estão localizados.

Possíveis respostas:

```
+SPOP OK
...
<diálogo entre o cliente e o servidor de autenticação>
...
```

`SPOP DONE!` – quando o servidor SPOP reconhecer o referido servidor de autenticação e o processo de autenticação foi concluído com sucesso.

`-SPOP ERR` – quando o servidor não for reconhecido, não puder ser encontrado, não estiver disponível ou não for possível estabelecer a conexão com o mesmo.

```
+SPOP OK
...
<diálogo entre o cliente e o servidor de autenticação>
...
```

`SPOP FAILED!` – quando ocorrer um erro durante a autenticação do usuário, tanto por motivo de erro nas informações fornecidas pelo cliente, quanto por falhas na comunicação com o servidor.

`+SPOP BREAK OFF` – quando o processo de autenticação for interrompido.

Exemplos:

```
C: AUTHSERVER CyberSafe 143.80.100.16:2185
S: +SPOP OK
...
C: *
S: +SPOP BREAK OFF
C: AUTHSERVER KyberPASS 143.80.100.16:2163
S: -SPOP ERR
C: AUTHSERVER SafeWord 143.80.100.16:2173
S: +SPOP OK
...
S: SPOP DONE!
```

Discussão sobre o comando:

O comando AUTHSERVER foi idealizado para flexibilizar o aspecto de segurança do servidor SPOP, proporcionando a utilização de servidores de autenticação independentes. O cliente SPOP se desejar que a autenticação do usuário seja mais segura, deverá implementar as chamadas ao servidor de autenticação, neste caso o servidor SPOP apenas serve de intermediário entre as partes durante a negociação.

A negociação entre o cliente e o servidor de autenticação é interrompida pelo SPOP, quando o cliente envia uma linha contendo apenas um asterisco ("*").

A diferença básica entre os comandos AUTHENTICATE e AUTHSERVER é que o mecanismo de autenticação referido pelo AUTHENTICATE deve fazer parte e é implementado pelo servidor SPOP. Já no AUTHSERVER o mecanismo de autenticação está fora do SPOP e tem de ser acessado através de uma interface de cliente. Para maior segurança do sistema, a interface cliente para o servidor de autenticação tem de ser conhecida pelo cliente que solicitou a autenticação especial, o servidor SPOP intermedia a negociação e reconhece quando o servidor de autenticação conclui a negociação com o usuário.

Este comando foi idealizado para explorar as funcionalidades de servidores próprios para a autenticação de usuários, como o SafeWord [Bosen96], KyberPASS [Kyber96], CyberSafe [Cyber98], OpenVision [Open98], SiteMinder [Banya98], Interceptor [Compu97] e outros de eficiência garantida, encontrados entre as ferramentas de segurança disponíveis no mercado.

Um servidor de autenticação independente, como o exemplo do SafeWord é utilizado através de sua API, que é a interface que o cliente vai utilizar para autenticar-se. Os servidores de autenticação implementam protocolos de rede que são utilizados para confirmar a identidade do usuário, tais como: RADIUS [Aboba99] [Zorn99], TACACS+ [Ander84], EASSP [Johns85] e Extended TACACS [Borma93].

II. Comandos para encerrar a conexão

a) END

Sintaxe:

```
END
```

Argumentos:

O comando END não possui argumentos.

Restrições:

O comando END não possui nenhuma restrição, pode ser utilizado a qualquer momento da conexão.

Possíveis respostas:

`+SPOP OK` - quando o servidor estiver pronto para encerrar a conexão.

`+SPOP STILL ANSWERS [rótulos] CONFIRM?` – quando existirem comandos a serem respondidos, para que o servidor encerre a conexão será necessária uma confirmação do cliente.

`-SPOP ERR` – quando algum erro impedir o encerramento da conexão, o cliente deve tentar novamente algumas vezes, persistindo o erro pode encerrar unilateralmente a conexão.

Exemplos:

```
C: END
S: +SPOP OK
S: +SPOP SAYS BYE, BYE
```

Discussão sobre o comando:

O comando END pode ser utilizado a qualquer momento da sessão para interromper a conexão com o servidor. Antes de encerrar propriamente a sessão com o cliente o servidor faz a atualização da caixa de entrada do usuário, bem como de todas as pastas da caixa de correio que estiverem abertas. Durante a atualização da caixa de correio e servidor apaga as mensagens marcadas com o sinalizador "/excluir". Este sinalizador é colocado pelo comando REMOVE, que é explicado com detalhes mais a frente.

b) CONFIRM

Sintaxe:

`CONFIRM`

Argumentos:

O comando CONFIRM não possui argumentos.

Restrições:

O comando CONFIRM só tem sentido de ser usado após um comando END cuja resposta indicou a necessidade de uma confirmação para que a conexão possa ser encerrada de fato. Pode ser utilizado em qualquer estágio da conexão, desde que

satisfazendo a sua finalidade. O comando CONFIRM utilizado aleatoriamente não tem funcionalidade nenhuma.

Possíveis respostas:

+SPOP OK – quando o servidor estiver pronto para encerrar a conexão.

-SPOP ERR – quando algum erro impedir o encerramento da conexão, o cliente deve tentar novamente algumas vezes, persistindo o erro pode encerrar unilateralmente a conexão.

Exemplos:

```
C: END
S: +SPOP STILL ANSWRES CB026 CB030 CB035 CONFIRM?
C: CONFIRM
S: +SPOP OK
S: +SPOP SAYS BYE, BYE
```

Discussão sobre o comando:

O comando CONFIRM tem por sua funcionalidade básica confirmar o encerramento de uma conexão entre um servidor e um cliente SPOP, enquanto ainda existem comandos do cliente que ainda não foram respondidos. Quando o cliente SPOP sinaliza com o CONFIRM ele está informando ao servidor que as respostas aos comandos que ele próprio enviou não são mais necessárias.

III. Comando de controle de conexão

a) PING

Sintaxe:

```
PING
```

Argumentos:

O comando PING não possui argumentos.

Restrições:

O comando PING não possui restrições, pode ser utilizado a qualquer momento da conexão.

Possíveis respostas:

+SPOP PONG OK (X ms) - quando servidor estiver funcionando normalmente.

+SPOP PONG OVERLOAD - quando o servidor estiver sobrecarregado.

Exemplos:

```
C: PING
S: +SPOP PONG OK
C: PING
S: +SPOP PONG OVERLOAD
```

Discussão sobre o comando:

O comando PING tem como funcionalidade básica possibilitar ao cliente SPOP averiguar se o servidor SPOP está funcionando normalmente, ou está sobrecarregado, ou simplesmente não está funcionando. Todo cliente SPOP deve ter internamente um controlador do tempo de resposta do servidor ao comando PING, caso esse tempo ultrapasse um limite predeterminado a conexão deve ser encerrada pelo cliente. É sugerido que o limite de tempo para a espera do cliente por uma resposta do servidor deva ser um parâmetro configurável, para que possa ser ajustado a condição atual do sistema. A escolha deste limite é de responsabilidade da administração o cliente SPOP. Também deve ser controlado pelo cliente o tempo que a mensagem levou para atingir o servidor e voltar ao cliente, este valor indicará momentos de sobrecarga no servidor. Quando não houver resposta, ou simplesmente se ultrapassou o limite de tempo para esta, o cliente deve proceder o encerramento da conexão. A ausência de resposta indica que o servidor não está funcionando adequadamente.

3.2.2.2. Comandos permitidos nos estágios "AUTENTICADO" e "SELECIONADO"

Além dos comandos: PING (controle) e END (encerramento), outros comandos são permitidos apenas quando o cliente já tem acesso a sua caixa de correio. Esses comandos são usados pelo cliente para que ele possa explorar, criar, apagar, renomear e duplicar pastas de mensagens.

I. Comando de controle de caixa de correio

a) STATUS

Sintaxe:

```
STATUS
```

Argumentos:

O comando STATUS não possui argumentos.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP FF MM NN ROOT MM NN` - exibe o número de pastas, mensagens e novas mensagens da pasta que está sendo explorada. Após o indicador ROOT o comando oferece detalhes gerais sobre a caixa de correio como o número de mensagens total da caixa de correio e o número de novas mensagens.

`-SPOP ERR` - quando estiver em um estágio não permitido para o comando.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
S: +SPOP OK, WELCOME
C: STATUS
S: +SPOP 5 30 4 ROOT 5 2
```

Discussão sobre o comando:

Com o comando STATUS o cliente busca informações sobre um caixa de correio específica. No exemplo acima a resposta do servidor ao comando STATUS do cliente indica que a referida caixa de correio possui ao todo 5 pastas de arquivamento, 30 mensagens e 4 novas mensagens (com o flag "\Recent" ou "\Unseen"). Além disso informa que na caixa de entrada (*INBOX*) existem 5 mensagens, e dentre elas existem 2 novas mensagens.

A resposta ao comando STATUS tem duas conotações. Caso o cliente não tenha aberto nenhuma pasta para trabalhar, as informações obtidas como resposta do comando STATUS dizem respeito a caixa de correio como todo e a caixa de entrada. Se uma pasta estiver aberta a resposta do comando STATUS dirá respeito a todo o subnível compreendido por aquela pasta, além das informações sobre as suas mensagens.

O comando STATUS de nenhuma forma altera o estado de uma caixa de correio e consequentemente o estado de nenhuma mensagem (seus flags, por exemplo).

II. Comandos de manipulação de pastas e mensagens

a) CREATE

Sintaxe:

`CREATE [TRASH] <nome da pasta> <regra de formação>`

Argumentos:

São três os possíveis argumentos passados para o comando CREATE, o primeiro argumento é opcional e indica se a pasta terá o atributo especial TRASH. O segundo é o nome da pasta de arquivamento de mensagens que se deseja criar. O terceiro argumento é a regra de formação que vai especificar as características das mensagens que serão vinculadas a essa pasta.

Restrições:

Só pode ser executado nos estágios "AUTENTICADO" ou "SELECIONADO" da sessão, além de que o nome da pasta a ser criada não pode ser um nome já existente.

Possíveis respostas:

`+SPOP OK` – quando a pasta for criada com sucesso.

`-SPOP ERR` – quando não puder ser criada uma pasta com o nome informado, o nome já existe ou a operação é ilegal no estágio atual da sessão com o servidor.

`-SPOP FAILED` – quando ocorrer um erro durante a criação da pasta.

Exemplos:

```
C: CREATE CNPq SUBJECT WITH "[CNPq-l]"
S: +SPOP OK
C: CREATE TRASH lixo BODY WITH "OFF TOPIC"
S: +SPOP OK
```

Discussão sobre o comando:

A pasta criada com o atributo especial TRASH terá o seu conteúdo excluído cada vez que se encerrar uma sessão envolvendo a referida caixa postal eletrônica. A linguagem na qual são escritas as regras de formação das pastas de arquivamento de mensagens chama-se SFDL. A linguagem SFDL será definida na sessão 3.2.3. Algumas observações se fazem necessárias neste momento, uma delas diz respeito aos nomes das pastas a serem criadas; eles obedecerão a uma estrutura hierárquica. Quando da criação de uma pasta que estará em um subnível de uma pasta já existente, ao se informar o nome da pasta a ser criada deve-se incluir também as pastas de hierarquia inferior. Como, por exemplo, ao se criar uma pasta "família" que está no subnível da pasta "pessoais", deve-se referir-se a ela no momento da criação como "pessoais/família".

Mais exemplos da criação de pastas podem ser encontrados na sessão 3.2.3.2.

b) RENAME

Sintaxe:

```
RENAME <nome original> <novo nome da pasta>
```

Argumentos:

Serão submetidos como parâmetros do comando RENAME o nome de origem da pasta, bem como o novo nome da mesma.

Restrições:

Só poderá ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão. O nome da pasta que se deseja renomear tem de existir previamente, uma nova pasta não é criada com o comando RENAME.

Possíveis respostas:

`+SPOP OK` – quando a pasta for renomeada com sucesso.

`-SPOP ERR` – quando o nome da pasta que se deseja alterar não existe, não é permitido a alteração do nome da referida pasta, ela está em uso ou o comando é ilegal no atual estágio da sessão com o servidor.

`-SPOP FAILED` – quando o processo de mudança de nome de uma pasta não conseguiu chegar ao seu final.

Exemplos:

```
C: RENAME Pessoais Antigas
S: +SPOP OK
C: RENAME Pessoais/Família Pessoais/Arquivo
S: -SPOP ERR
C: RENAME Antigas/Família Arquivo
S: +SPOP OK
```

Discussão sobre o comando:

O comando RENAME tem a funcionalidade de modificar o nome das pastas de vinculação de mensagens de uma caixa de correio. Pode ser utilizado ainda para mudar o nível da pasta na hierarquia das pastas existentes. Como foi ilustrado no exemplo acima, a pasta "Família" que estava dentro da pasta "Antigas" teve não só o seu nome alterado mas passou ao mesmo nível da pasta "Antigas".

A alteração do nome de uma pasta não afeta em nada a sua constituição, que é definida pela regra de formação da mesma. O que significa que as regras de formação de pastas não são herdadas por suas subpastas, mantendo desta forma

uma independência entre as pastas. A independência entre as pastas deixa o trabalho com estas mais flexível, mas a estrutura de composição de regras de formação por hierarquia deve ser motivo de estudos futuros sobre o assunto.

O comando RENAME é uma ferramenta poderosa para administração das pastas de arquivamentos de mensagens de uma caixa de correio. Com ela um cliente SPOP pode reorganizar toda a árvore das pastas, bem como alterar os seus nomes.

c) SHOW

Sintaxe:

```
SHOW [<pasta> | <mensagem>]
```

Argumentos:

O comando SHOW pode ser utilizado sem argumentos quando se tratar das informações da pasta correntemente utilizado. Este comando pode ainda utilizar uma de duas opções, ou o nome de uma pasta, ou o número de uma mensagem.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

```
+SPOP MM MESSAGES (BB BYTES)
/pasta 1
/pasta n
1 <número de bytes>
N <número de bytes>
```

SPOP END SHOW! - exibe as informações da pasta em questão.

-SPOP ERR - quando estiver em um estágio não permitido para o comando ou não existir a referida pasta.

-SPOP FAILED - quando não for possível realizar o comando.

Exemplos:

```
C: SHOW 3
S: +SPOP 6 MESSAGES (893 BYTES)
S: 3 120
S: SPOP END SHOW!
C: SHOW
S: +SPOP 6 MESSAGES (893 BYTES)
S: /Universidade
S: /Pessoais
```

```
S: 1 103
S: 2 90
S: 3 120
S: 4 94
S: 5 220
S: 6 266
S: SPOP END SHOW!
C: SHOW Pessoais
S: +SPOP 2 MESSAGES (110 BYTES)
S: /Pessoais/Cartas
S: 10 75
S: 11 35
S: SPOP END SHOW!
S: SHOW Pessoais/Receitas
S: -SPOP ERR
```

Discussão sobre o comando:

O comando SHOW, como o próprio nome já diz, mostra a lista das subpastas e mensagens contidas na pasta atualmente em uso, exibindo ao cliente os números seqüenciais destas mensagens e o seu tamanho em bytes. Este comando pode ser de uso restritivo ou geral. É de uso restritivo quando o cliente informa o número que representa uma mensagem específica ou quando ele fornece o nome de uma pasta de correio para recuperar as suas informações. A outra opção de uso do comando é de forma geral, onde serão exibidas as informações da pasta que estiver aberta, caso nenhuma pasta esteja aberta as informações mostradas serão das mensagens da caixa de entrada.

Como a resposta do comando SHOW utiliza várias linhas, o servidor SPOP não enviará outra resposta até que todas as linhas que compõem esta resposta sejam recebidas pelo cliente. Está precaução evitará confusões nas respostas recebidas pelo cliente.

A lista com as informações das mensagens possui um formato simples, cada linha após o início da resposta representa uma pasta ou uma mensagem. As linhas que representam pastas de vinculação de mensagens iniciarão com um ponto, seguido de uma barra e pelo nome da pasta. Nas linhas que representam mensagens constarão o número da mensagem e o seu tamanho em bytes separados por um espaço em branco e seguidos do caracter especial de nova linha.

Uma vez o cliente explorando qualquer das pastas de arquivamento de mensagens a qualquer subnível pode fazer referência à caixa de entrada utilizando-se do caractere barra ("/"), este elemento simboliza a raiz de todas as pastas do usuário. Portanto, se for utilizado o comando "`SHOW /`", o resultado refletirá as informações contidas na caixa de entrada do usuário.

Os nomes das pastas à medida que forem descendo nos subníveis vão incorporando ao seu, os nomes das pastas que estão nos níveis superiores, as quais chamaremos de pastas ancestrais. Como no exemplo mostrado, a pasta "Pessoais" tem uma subpasta, que por sua vez chamar-se-á "Pessoais/Cartas".

d) FETCH

Sintaxe:

```
FETCH <número da mensagem> [HEADER | TEXT | ALL]
```

Argumentos:

O argumento principal, e obrigatório, para este comando é o número da mensagem que se deseja recuperar. O segundo argumento é opcional e indica como deve acontecer a recuperação da referida mensagem.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

```
+SPOP OK MESSAGE MM HAVE BB BYTES

... (mensagem propriamente dita)
```

SPOP END! - quando a mensagem requisitada existe.

-SPOP ERR - quando não existir a referida mensagem na pasta em uso.

-SPOP FAILED - quando não for possível realizar o comando.

Exemplos:

```
C: FETCH 2
S: +SPOP OK MESSAGE 2 HAVE 544 BYTES
S: From fellipe@dsc.ufpb.br Tue Jan 26 14:29:28 1999
S: Date: Tue, 26 Jan 1999 14:28:39 -0300 (GRNLNDST)
S: From: "Fellipe Araujo Aleixo / (Pos-Graduacao DSC)"
   <fellipe@dsc.ufpb.br>
S: To: lsd@dsc.ufpb.br
S: Subject: Como o spop recupera uma mensagem.
S:
S: Olah LSD,
S: ...
S: SPOP END!
C: FETCH 100
S: -SPOP ERR
```

Discussão sobre o comando:

O comando FETCH é utilizado pelo cliente SPOP para recuperar mensagens específicas. Após autenticar-se o cliente já tem acesso às mensagens que estão localizadas na caixa de entrada da referida caixa de correio. Se o servidor SPOP iniciar uma resposta positivamente indicará quantos bytes, correspondentes à mensagem, serão transferidos.

O servidor SPOP provê a opção em que o cliente SPOP especifica como deve se dá a recuperação da mensagem especificada. As opções são: "HEADER", que recupera apenas o cabeçalho da mensagem; "TEXT", que recupera apenas as partes das mensagens no formato texto (*text/plain*), até mesmo em caso de mensagens com várias partes (MIME); e "ALL", que é a opção padrão, onde a mensagem completa é recuperada.

e) EXAMINE

Sintaxe:

```
EXAMINE <nome da pasta> [HEADER | TEXT | ALL]
```

Argumentos:

O argumento principal, e obrigatório, para este comando é o nome da pasta de vinculação que se deseja recuperar. O segundo argumento é opcional e indica como deve acontecer a recuperação das mensagens vinculadas à referida pasta.

Restrições:

Pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

```
+SPOP OK, OPENING <nome da pasta>
+SPOP MESSAGE MM HAS BB BYTES
... (mensagem 1 propriamente dita)
+SPOP MESSAGE MM HAS BB BYTES
... (mensagem 2 propriamente dita)
...
```

`SPOP END!` - quando a pasta referida existe e as mensagens foram recuperadas com sucesso.

`-SPOP ERR` - quando não existir a referida pasta.

-SPOP FAILED - quando o comando não conseguir ser completado.

Exemplos:

```
C: EXAMINE Pessoais/Arquivo
S: +SPOP OK, OPENING /Pessoais/Arquivo
S: +SPOP MESSAGE 10 HAVE 544 BYTES
S: From fellipe@dsc.ufpb.br Tue Jan 26 14:29:28 1999
S: Date: Tue, 26 Jan 1999 14:28:39 -0300 (GRNLNDST)
S:  From: "Fellipe Araujo Aleixo / (Pos-Graduacao DSC)"
   <fellipe@dsc.ufpb.br>
S: To: lsd@dsc.ufpb.br
S: Subject: Como o spop recupera uma mensagem.
S:
S: Olah LSD,
S: ...
S: +SPOP MESSAGE 27 HAVE 925 BYTES
S: Received: from lsd.dsc.ufpb.br [150.165.85.5] by
   lsd.dsc.ufpb.br [150.165.85.5] with RAW
   (MDaemon.v2.7.SP5.R) for <fellipe@lsd.dsc.ufpb.br>;
   Wed, 07 Apr 1999 11:31:51 -0300
S: Date: Wed, 07 Apr 1999 11:31:51 -0300
S: From: MDaemon@lsd.dsc.ufpb.br
S: X-MDSend-Notifications-To: [trash]
S: Subject: Mailing list listing
S: To: fellipe@lsd.dsc.ufpb.br
S: X-MDaemon-Deliver-To: fellipe@lsd.dsc.ufpb.br
S: Reply-To: BadMsgQ@lsd.dsc.ufpb.br
S: Message-ID:
   <MDAEMON770077199904071131.AA3151049@lsd.dsc.ufpb.br>
S: Mime-Version: 1.0
S: Content-Type: text/plain; charset=US-ASCII
S: X-Actual-From: MDaemon@lsd.dsc.ufpb.br
S:
S: LIST command recognized
S: ...
S: SPOP END!
C: FETCH Arquivo/Pessoais
S: -SPOP ERR
```

Discussão sobre o comando:

O comando EXAMINE tem o efeito semelhante ao comando FETCH, e é utilizado pelo cliente SPOP para recuperar as mensagens vinculadas a um SFolder. O comando EXAMINE torna a recuperação de mensagens mais simples, pois minimiza as interações entre o servidor e o cliente SPOP, quando este último já sabe exatamente o que deseja recuperar.

O servidor SPOP provê a opção em que o cliente SPOP especifica como deve se dar a recuperação das mensagens vinculadas ao SFolder. As opções são: "HEADER", que recupera apenas o cabeçalho da mensagem; "TEXT", que recupera apenas as partes das mensagens no formato texto (*text/plain*), até mesmo em caso de

mensagens com várias partes (MIME); e "ALL", que é a opção padrão, onde a mensagem completa é recuperada.

f) HEADER

Sintaxe:

```
HEADER <mensagens> <número de linhas>
```

Argumentos:

Os argumentos para este comando são os números das mensagens e o número de linhas destas que se deseja recuperar.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

```
+SPOP OK MESSAGE MM HAVE BB BYTES
...
SPOP END!
```

- A primeira linha é utilizada quando a mensagem requisitada existe e vai ser iniciada a transferência, a última linha indica que a transmissão da mensagem foi concluída.

`-SPOP ERR` – quando não existir a referida mensagem na pasta em uso.

`-SPOP FAILED` - quando a mensagem existe, mas não foi possível completar o comando.

Exemplos:

```
C: HEADER 2 2
S: +SPOP OK MESSAGE 2
S: From fellipe@dsc.ufpb.br Tue Jan 26 14:29:28 1999
S: Date: Tue, 26 Jan 1999 14:28:39 -0300 (GRNLNDST)
S: +SPOP END
C: HEADER 2 5
S: From fellipe@dsc.ufpb.br Tue Jan 26 14:29:28 1999
S: Date: Tue, 26 Jan 1999 14:28:39 -0300 (GRNLNDST)
S:  From: "Fellipe Araujo Aleixo / (Pos-Graduacao DSC)"
   <fellipe@dsc.ufpb.br>
S: To: lsd@dsc.ufpb.br
S: Subject: Como o SPOP recupera uma mensagem.
S: SPOP END!
C: HEADER 100
S: -SPOP ERR
```

Discussão sobre o comando:

O comando HEADER é utilizado para se recuperar cabeçalhos de mensagens de correio. Geralmente é utilizado para minimizar o esforço de transferência do cliente SPOP, neste caso ele pode optar por transferir apenas as informações dos envelopes de cada mensagem e então optar por quais mensagens ele deseja recuperar por inteiro.

Os argumentos do comando HEADER são simples e objetivos, indicam a mensagem desejada e o número de linhas a partir do início da mensagem que se deseja recuperar, ambos separados por espaços em branco.

g) REMOVE

Sintaxe:

```
REMOVE <mensagens> [NOW]
```

Argumentos:

O comando REMOVE é utilizado para remover mensagens, os argumentos para este comando são os números das mensagens que se deseja apagar separados por um espaço em branco e opcionalmente a indicação "NOW".

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK <indicador> DELETED` - quando a mensagem requisitada existe e foi marcada para ser apagada.

`+SPOP OK <indicador> DELETED NOW` - quando a mensagem requisitada existe e foi apagada, de modo que não pode ser recuperada.

`-SPOP ERR MM NON EXISTENT` - quando a mensagem referida não existe.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: REMOVE 6 NOW
S: +SPOP OK 6 DELETED NOW
C: REMOVE 19 20 21 22
S: -SPOP ERR 22 NON EXISTENT
S: -SPOP ERR 21 NON EXISTENT
S: +SPOP OK 20 DELETED
```

```
S: +SPOP OK 19 DELETED
```

Discussão sobre o comando:

O comando REMOVE é utilizado pelo cliente SPOP para marcar as mensagens do usuário corrente que serão descartadas quando a conexão com o cliente for. Ainda utilizando o comando REMOVE o servidor pode descartar definitivamente mensagens, fazendo uso da opção "NOW" após as referências às mensagens. Este comando pode ser utilizado para apagar uma mensagem específica ou um conjunto delas.

Quando existem várias mensagens a serem apagadas o cliente SPOP pode fazer um só pedido para que todo este conjunto seja apagado. Os números das mensagens que se deseja apagar devem seguir o comando REMOVE e devem estar separadas por espaços em branco.

O comando REMOVE utiliza uma regra para apagar uma série de mensagens, ele sempre apaga as mensagens na ordem decrescente dos seus números de referência. Como no exemplo acima, se o cliente tem uma lista de quatro mensagens para apagar com um só comando REMOVE, não importa a ordem que ele informe ao servidor, as mensagens serão excluídas na prática na ordem decrescente.

h) DELFOLDER

Sintaxe:

```
DELFOLDER <pastas> [NODELMSG]
```

Argumentos:

O comando DELFOLDER é utilizado para remover pastas. Para remover pastas, os argumentos são os nomes das pastas que se deseja remover separados por espaços em branco e opcionalmente o complemento "NODELMSG".

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK <indicador> DELETED` - quando a pasta requisitada existe e foi removida.

`-SPOP ERR <nome da pasta> NON EXISTENT` - quando a referida pasta não existir.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: DELFOLDER Pessoais/Cartas Pessoais/Convites NODELMSG
S: +SPOP OK Pessoais/Cartas DELETED
S: -SPOP ERR Pessoais/Convites NON EXISTENT
```

Discussão sobre o comando:

O comando DELFOLDER é utilizado pelo cliente SPOP para apagar pastas de vinculação de mensagens. Este comando pode ser utilizado para apagar um conjunto de pastas. Quando existem várias pastas a serem apagadas o cliente SPOP pode fazer um só pedido para que todo este conjunto seja apagado. Os nomes das pastas que se deseja apagar devem seguir o comando DELFOLDER e devem estar separados por espaços em branco.

Quando o cliente está fazendo um pedido para que o servidor apague uma pasta de arquivamento de mensagens ele pode fazer uso da opção "NODELMSG", ela informa ao servidor que as mensagens contidas na pasta não devem ser apagadas. As mensagens vinculadas a uma pasta que está sendo removida através do comando DELFOLDER com a opção "NODELMSG", não são apagadas. Se estas mensagens não estiverem vinculadas a outras pastas, serão novamente vinculadas a caixa de entrada.

Por motivos de segurança, o comando DELFOLDER não apaga uma hierarquia de pastas, ou seja, o servidor não pode apagar pastas que possuam subpastas dentro delas. Desta forma, se o cliente desejar apagar uma pasta qualquer terá que eliminar primeiro todas a subpastas desta para só então apagar a mesma. Outro detalhe, o comando DELFOLDER não pode apagar a caixa de entrada (representada por "/").

i) UNMARK

Sintaxe:

```
UNMARK <mensagens>
```

Argumentos:

Os argumentos para este comando são os números das mensagens que se deseja remover as marcas deixadas pelo comando REMOVE.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

+SPOP OK <indicador> UNMARKED - quando a mensagem existe e as marcas deixadas pelo comando REMOVE foram retiradas.

+SPOP MM ALREADY UNMARKED - quando a mensagem referida não possuir a marca para ser apagada.

-SPOP ERR MM DOES NOT EXIST - quando a mensagem referida não existe.

-SPOP FAILED - quando não for possível realizar o comando.

Exemplos:

```
C: UNMARK 18 19 20 21 22
S: +SPOP 18 ALREADY UNMARKED
S: +SPOP OK 19 UMARKED
S: +SPOP OK 20 UMARKED
S: -SPOP ERR 21 DOES NOT EXIST
S: -SPOP ERR 22 DOES NOT EXIST
```

Discussão sobre o comando:

O comando UNMARK é utilizado para se recuperar uma mensagem que foi apagada com o comando REMOVE. Com este comando só é possível recuperar as mensagens marcadas para serem removidas na sessão corrente. Geralmente ele se faz necessário para reparar erros ou mudanças de atitude do usuário e não penaliza-lo por apagar uma mensagem por engano.

Este comando pode ser utilizado para remover as marcas do comando REMOVE em várias mensagens ao mesmo tempo. Para que isso aconteça os números das mensagens que se deseja recuperar devem seguir o comando e devem estar separados por espaços em branco.

j) UIDM

Sintaxe:

```
UIDM <mensagens>
```

Argumentos:

O argumento para este comando são os números das mensagens das quais se deseja recuperar os identificadores únicos de cada mensagem.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK MESSAGE MM HAVE UID UU` - quando a referida mensagem existe.

`-SPOP ERR MM UID` - quando não existir a mensagem na pasta em uso.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: UIDM 2 3 4 5
S: +SPOP OK MESSAGE 2 HAVE UID 2385308
S: +SPOP OK MESSAGE 3 HAVE UID 2618434
S: -SPOP ERR 4 UID
S: -SPOP ERR 5 UID
```

Discussão sobre o comando:

O comando UIDM é utilizado para a recuperação dos identificadores únicos de um conjunto de mensagens. Os identificadores únicos são números que identificam univocamente cada mensagem em uma caixa de correio. Os identificadores são compostos de números não necessariamente seqüenciais e geralmente crescentes. São formados de números grandes para suportar uma grande quantidade de mensagens. O mecanismo de ordenação o UID das mensagens tem por finalidade básica a diferenciação das mensagens que estiveram na caixa de correio. Uma vez a caixa de correio vazia o UIDM pode ser reiniciado.

Pode ser feito o pedido dos UIDM de várias mensagens, para fazer isto basta seguir o comando com o conjunto dos números das mensagens, separados por espaços em branco. A resposta do servidor será individual para cada mensagem, portanto não haverá problemas se algumas das mensagens em questão não existirem a resposta para as outras mensagens não será comprometida.

l) TRIGGER

Sintaxe:

`TRIGGER <nome> [ON, WHEN <regra de formação> DO <ação>|OFF]`

Argumentos:

O comando TRIGGER é utilizado para ativar e desativar condições de disparo para ações. Para ativação são necessários três argumentos, o primeiro é o nome da condição de disparo. O segundo é uma regra de formação que especifica um determinado critério para ser procurado entre as mensagens do usuário. O terceiro argumento é a indicação de uma ação a ser realizada, essas ações podem ser de dois

tipos, ou *scripts* [4]em SFDL ou ações padrão, já pré definidas. As ações até então definidas são:

```
CREATE SFOLDER "<string>" WITH THEM

DELETE THEM
```

Para desativação de uma condição de disparo, configurada anteriormente, basta apenas um argumento, que é o nome atribuído quando a condição de disparo foi ativada.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK, TRIGGER <nome> ON` - quando a condição de disparo for ativada.

`+SPOP OK, TRIGGER <nome> OFF` - quando a mesma for desativada.

`-SPOP ERR, BAD TRIGGER <nome>` - quando não existir a referida condição de disparo.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: TRIGGER pessoal ON, WHEN BODY WITH "Querido"
   DO CREATE SFOLDER "Pessoal" WITH THEM
S: +SPOP OK, TRIGGER pessoal ON
C: TRIGGER amigos OFF
S: -SPOP ERR, BAD TRIGGER amigos
C: TRIGGER pessoal OFF
S: +SPOP OK, TRIGGER pessoal OFF
```

Discussão sobre o comando:

O comando TRIGGER é um comando criado para aumentar a flexibilidade do SPOP. Ele é responsável pela ativação e desativação de regras de disparo de ações. Este comando associa ao sucesso na localização de um determinado critério nas mensagens da caixa de correio do usuário, a execução de uma determinada ação. A princípio definimos duas ações padrão que podem ser realizadas quando forem encontradas mensagens com determinadas características: 1) criação de um *SFolder*

[4] Conjunto de comandos em uma determinada linguagem, especificando um roteiro a ações a serem tomadas.

contendo como regra de formação a regra que foi utilizada na pesquisa, ou seja, contendo todas as mensagens que foram "selecionadas" pela regra; e 2) marcar para remoção as mensagens que satisfizeram a regra. A outra opção em termos de ação a ser executada, é a utilização de um *script* contendo uma série de comandos em SFDL. Esta opção oferece uma maior liberdade e flexibilidade ao cliente.

m) SELECT

Sintaxe:

```
SELECT <pasta>
```

Argumentos:

O argumento para este comando é a pasta que deseja-se selecionar.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK NOW IN <nome da pata>` - quando a referida pasta existe e está pronta para ser explorada.

`-SPOP ERR NO <nome da pasta>` - quando não existir a pasta requisitada.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: SELECT Bilhetes
S: -SPOP ERR NO Bilhetes
C: SELECT Pessoais
S: +SPOP OK NOW IN Pessoais
C: SELECT Bilhetes
S: +SPOP OK NOW IN Bilhetes
ou
C: SELECT /Pessoais/Bilhetes
S: +SPOP OK NOW IN /Pessoais/Bilhetes
```

Discussão sobre o comando:

O comando SELECT é utilizado para permitir ao cliente SPOP selecionar uma pasta específica para o trabalho. A identificação das pastas pode se dar de maneira direta (apenas quando esta for uma subpasta da pasta atual) ou pode ser referenciada pelo seu caminho completo. O comando SELECT é útil quando por exemplo

desejamos criar uma subpasta em uma determinada pasta já existente, para tal temos que seleciona-la primeiro e logo após podemos criar uma nova pasta.

n) DESELECT

Sintaxe:

```
DESELECT [/]
```

Argumentos:

Este comando geralmente não possui argumentos, o que apenas indica que se deseja subir um nível na hierarquia de pastas. O argumento opcional para este comando é o caracter "/" que indica que o cliente deseja retornar à caixa de entrada, não importando em que parte da árvore de pastas ele se encontra.

Restrições:

Só pode ser utilizado nos estágios "AUTENTICADO" e "SELECIONADO" da sessão com o servidor.

Possíveis respostas:

`+SPOP OK NOW IN <nome da pata>` - quando o comando for bem sucedido.

`-SPOP ERR ALREADY IN /` - quando o cliente executar o comando a partir da caixa de entrada.

`-SPOP FAILED` - quando não for possível realizar o comando.

Exemplos:

```
C: SELECT /Pessoais/Bilhetes
S: +SPOP OK NOW IN /Pessoais/Bilhetes
C: DESELECT
S: +SPOP OK NOW IN /Pessoais
C: SELECT Bilhetes
S: +SPOP OK NOW IN Bilhetes
C: DESELECT /
S: +SPOP OK NOW IN /
C: DESELECT
S: -SPOP ERR ALREADY IN /
```

Discussão sobre o comando:

O comando DESELECT é utilizado para permitir ao cliente SPOP voltar um nível na hierarquia de pasta enquanto o mesmo estiver navegando por ela. Permite também que o cliente retorne de qualquer ponto da estrutura de árvore das pastas de arquivamento para a caixa de entrada.

3.2.2.3. Comandos permitidos no estágio "ENCERRANDO"

Este estágio é iniciado quando o servidor SPOP recebe um comando END do cliente. Não são permitidos comandos no estágio "ENCERRANDO" da conexão, pois este momento é utilizado pelo servidor para fazer as últimas alterações na caixa postal do usuário e por fim, enviar um sinal ao cliente SPOP que está pronto para encerrar a conexão.

3.2.3. Linguagem SFDL

Smart Folder Definition Language (SFDL) é o nome dado à linguagem criada para possibilitar a especificação das regras de formação das pastas de arquivamento de mensagens. Estas regras especificam as características que as mensagens que serão encaminhadas a determinadas pastas deverão ter.

Existe apenas uma linguagem correlata, fruto de um trabalho ainda em andamento. Esta linguagem está sendo desenvolvida na Universidade Carnegie Mellon por T. Showalter e chama-se *Sieve* [Showa98]. A Sieve é uma linguagem para a filtragem de mensagens de correio eletrônico no momento da entrega final, que se dá entre o servidor de caixa de correio (MS) e o programa cliente que o usuário está usando (UA). Ela é desenvolvida para ser independente de protocolos, e pode tanto ser implementada no cliente como no servidor.

A Sieve tem como alvo no seu desenvolvimento algumas características: ser extensível; simples; e ser independente de protocolo de acesso, arquitetura de correio e sistema operacional.

Outro mecanismo existente que possui alguma semelhança com o mecanismo proposto pela linguagem SFDL pode ser encontrado no comando SEARCH do IMAP, que é utilizado pelo cliente IMAP para levar ao servidor a efetuar uma busca na caixa de correio do usuário por mensagens obedecendo um determinado critério de busca. Uma vez de posse do resultado do comando SEARCH o usuário pode então recuperar as mensagens. Os critérios de busca consistem em uma ou mais chaves de busca que podem explorar todo o conteúdo de uma mensagem, seu cabeçalho ou seu corpo.

A proposta da linguagem SFDL traz uma abordagem diferente do problema, em relação às outras duas soluções citadas acima. A SFDL é uma linguagem de construção de regras, que servirão de argumento aos comandos CREATE e TRIGGER, para a criação de pastas de arquivamento de mensagens e para a definição de condições de disparo de ações. Estas regras fundamentarão a funcionalidade do SPOP denominada pastas automáticas.

O mecanismo das pastas automáticas do SPOP funciona da seguinte maneira: quando o cliente faz um pedido para a criação de uma pasta, o servidor SPOP faz uma pesquisa na caixa postal do cliente para encontrar mensagens que satisfaçam ao critério estipulado pelo cliente na regra de formação que foi passada como argumento do comando CREATE. As mensagens que satisfizerem ao critério do usuário serão vinculadas à pasta recém criada, e uma vez a pasta criada todas as novas mensagens que chegarem à caixa de correio do usuário serão analisadas para se verificar a existência de uma, ou mais, pastas próprias para a mesma. A mensagem que não se encaixar nos critérios de nenhuma das pastas automáticas criadas pelo usuário permanecerá na caixa de entrada.

Existe uma diferença básica entre a proposta do SPOP e as outras abordagens. Essa diferença está no fato da persistência do resultado de uma pesquisa. Essa persistência se dá na forma de uma pasta. O resultado de um comando SEARCH no IMAP é válido apenas no momento da sua execução. Com o cliente IMAP encerrando a sua sessão com o servidor, ele pode até salvar em disco a resposta que obteve do servidor, mas além de não ser esta uma solução distribuída (pois ela só pode ser utilizada pelo o computador a partir de onde foi realizada e arquivada) como sempre prega o IMAP, a consistência desta resposta não tem como ser garantida. A linguagem *Sieve*, por tratar-se apenas da especificação de uma linguagem genérica para a recuperação de mensagens eletrônicas, não prevê em sua definição nenhum tipo de persistência dos resultados.

O fato de toda a funcionalidade de busca de mensagens, que atendam a um determinado conjunto de critérios, ser implementada no servidor, favorece a criação de clientes SPOP mais simples. Outro ponto a ser beneficiado com a implementação da funcionalidade de busca no servidor é o volume de dados que trafega entre o servidor e o cliente, que é reduzido consideravelmente. Os UAs SPOP serão mais simples que os UAs que utilizam o POP, pois não terão que implementar nenhum algoritmo de busca para realizar a pesquisa localmente, como acontece com muitos clientes do POP nos dias de hoje. Os benefícios de se restringir a complexidade das buscas já foi descoberto pelo IMAP; o SPOP propõe um melhoramento nesta estrutura, tornando-a mais simples e flexível.

O volume de dados transferidos entre o cliente e o servidor é reduzido, pois diferentemente das outras soluções, as pastas de arquivamento de mensagens não precisam de estímulos do cliente para estarem sempre consistentes, e a recuperação se torna direta. A recuperação de mensagens específicas torna-se mais direta em comparação com a solução mais próxima (que é a do IMAP), onde o servidor tem de

realizar o comando SEARCH, esperar pela resposta e de acordo com a resposta recebida efetuar a transferência de mensagens desejadas. No SPOP, como já existe uma pasta composta por mensagens que atendam a um determinado conjunto de critérios, basta apenas um comando para que estas mensagens sejam recuperadas para o cliente. Quando temos uma situação em que mensagens que correspondam a uma determinada características são constantemente necessárias, a solução provida pelo SPOP oferece vantagens com relação às demais.

Para definirmos a nossa gramática para a construção de regras de formação nos baseamos em Sistemas de Recuperação de Informação Textual [Mead92]. Como o que desejamos recuperar é informação, necessitamos primeiro definir o que é informação. Informação para o nosso estudo, apresenta as seguintes características: 1) pode ser representada por um conjunto de símbolos, 2) tem alguma estrutura e 3) pode ser lida e entendida pelos usuários da mesma. Para efetuarmos a recuperação da informação precisamos definir uma linguagem especial para especificar que tipo de informação se quer recuperar. O processo de análise de uma declaração em uma linguagem e quebra desta em seus elementos constituintes ou regras é conhecido como *parsing*.

Uma das metas principais da linguagem SFDL é ser simples, ao mesmo tempo que formal, oferecendo muitos recursos a serem explorados pelos clientes SPOP. Outra das suas metas é ser flexível, possibilitando a sua extensão.

3.2.3.1. Gramática formal da linguagem SFDL

```
<regra> := Ø  ||  <critério>  ||
      <trecho de regra> <operador> <trecho de regra>  ||
      (<trecho de regra> <operador> <trecho de regra>)  ||
      (<trecho de regra>) <operador> (<trecho de regra>)


<operador> := "||"  ||  "&&"


<trecho de regra>  := <critério>  ||  NOT(<trecho de regra>)  ||
      <trecho de regra> <operador> <trecho de regra>  ||
      (<trecho de regra> <operador> <trecho de regra>)  ||
      (<trecho de regra>) <operador> (<trecho de regra>)


<critério> := <critério de texto>  ||
      <critério de flags>  ||
      <critério de data>  ||
      <critério numérico>  ||
      NOT(<critério>)
```

```
<critério de texto> := <atributo de texto> WITH "<string>"  ||
                       <atributo de texto> WITH <associação de texto>


<associação de texto> := "<string>" <associação> "<string>"   ||
                         PREFIX "<string>"   ||
                         SUFIX "<string>"


<associação> := SPHRASE      ||     //Mesma frase
                SPARAGRAPH   ||     //Mesmo parágrafo
                (<valor>)    ||     //Intervalo de proximidade
                (L<valor>)   ||     //Proximidade de localização
                (N<valor>)          //Adjacência


<string> := Ø  ||  <apenas texto>  ||
            <string> <expressão regular> <string>


<apenas texto> := <caracter>  ||  <caracter><string>


<caractere> := < qualquer caracter ASCII exceto <especiais> >


<especiais> := NULL  ||  "


<expressão regular> := .      ||    //Qualquer caracter
       \<expressão regular>   ||    //Escape
       [<sequência>]          ||    //Sequência
       [^<sequência>]         ||    //Não possui sequência
       #                      ||    //Qualquer número
       {<valor>-<valor>}      ||    //Intervalo
       <expressão regular>*   ||    //Repetição 0 ou mais vezes
       <expressão regular>+   ||    //Repetição 1 ou mais vezes
       <expressão regular>?   ||    //Opcional
     <expressão regular><expressão regular>   ||  //Justaposição
     <expressão regular>|<expressão regular>  ||  //OU


<sequência> := <apenas texto>  ||
               <caracter>-<caracter>  ||
               <numero>-<numero>

<atributo de texto> := BCC || BODY || CC || FROM || SUBJECT ||
      TEXT || TO


<critério de flags> := WITH FLAGS <lista de flags>


<lista de flags> := <flag>  ||  <flag> <lista de flags>
```

```
<flag> := \ANSWERED || \DELETED || \DRAFT || \FLAGGED ||
      \RECENT || \SEEN

<critério de data> := <ocorrência> <comparador> <data>

<ocorrência> := SENT DATE  ||  RECEIVE DATE

<comparador> := > || < || = || >= || <= || !=

<data> := <número><número>/<número><número>/
      <número><número><número><número>  ||
      <referencia de data> || <expressão de data>

<referencia de data> := <dia> || <mês> || <ano>

<dia> := SUNDAY || MONDAY || TUESDAY || WEDNESDAY || THURSDAY ||
      FRIDAY || SATURDAY || TODAY || YESTERDAY

<mês> := JANUARY || FEBRUARY || MARCH || APRIL || MAY || JUNE ||
      JULY || AUGUST || SEPTEMBER || OCTOBER || NOVEMBER ||
      DECEMBER || THISMONTH || LASTMONTH

<ano> := THISYEAR || LASTYEAR

<expressão de data> := <referencia de data> <operação> <valor>

<operação> := + || -

<número> := 0 || 1 || 2 || 3 || 4 || 5 || 6 || 7 || 8 || 9

<critério numérico> := <atributo numérico> <comparação>

<atributo numérico> := MESSAGE SET  ||  SIZE  ||  UID

<comparação> := <comparador> <valor>  ||  <intervalo>

<valor> := <número>  ||  <número><valor>

<intervalo> := IN <valor>..<valor>
```

3.2.3.2. Exemplos de utilização das regras de formação

Nesta seção mostraremos alguns exemplos de regras, que especificam uma série de critérios que mensagens de correio eletrônico devem conter para fazer parte de uma determinada seleção.

Exemplo 1:

Temos hoje em dia uma padronização das listas de discussão, que utilizam no início do campo que determina o assunto da mensagem o próprio nome da lista entre colchetes. Com base nesta afirmação que fizemos, para selecionar as mensagens recentes de uma lista denominada "java-l" que tragam algo sobre o assunto "threads", seria possível utilizar a seguinte regra:

```
WITH FLAGS \RECENT && SUBJECT WITH "[java-l]" && TEXT WITH
"thread"
```

Exemplo 2:

Para otimizar a utilização do espaço por uma determinada caixa postal eletrônica, o cliente pode especificar uma série de critérios para definir as mensagens que este não deseja manter na sua caixa postal eletrônica. Para tal basta criar uma pasta com o atributo especial TRASH e especificar os critérios das mensagens que deseja excluir. Por exemplo, temos um determinado usuário deseja que sejam excluídas de sua caixa postal eletrônica as mensagens provenientes do domínio "sales.com", pois é sabido que as mensagens provenientes deste domínio trazem apenas ofertas de compras pela Internet que não são interessantes ao usuário; também deseja que sejam excluídas mensagens com mais de 500 Kbytes, para que sua caixa postal eletrônica não venha a extrapolar a cota de espaço permitida; deseja excluir também mensagens de um usuário específico, como "intruso@dsc.ufpb.br"; além de mensagens no formato HTML:

```
FROM WITH "sales.com" || SIZE > 512000 ||
FROM WITH "intruso@dsc.ufpb.br" || BODY WITH "<HTML>"
```

Exemplo 3:

Suponhamos a situação de um alto executivo de uma grande empresa que possui um sistema de correio eletrônico para facilitar a comunicação interna entre seus funcionários. Como forma de melhorar a organização, cada departamento da empresa contém a sua própria sub-rede, o seu próprio domínio e o seu próprio servidor de correio eletrônico. Como este executivo trabalha supervisionando vários departamentos da empresa, ele poderia querer arquivar em uma pasta diferente as

mensagens provenientes dos funcionários de um determinado departamento da empresa. Este executivo poderia separar em uma pasta todas as mensagens assinaladas com a palavra "URGENTE". Poderia ainda marcar para excluir as mensagens com pedido de aumento. Poderia ainda, separar em uma pasta as mensagens relativas a um projeto específico no qual está trabalhando com mais três funcionários da empresa. Por fim, poderia separar por semestre os memorandos do ano de 1999. Para atender as necessidades deste alto executivo seria criado um conjunto de pastas com as seguintes regras:

Pasta do departamento de vendas:

```
FROM WITH "@vendas.empresa.com"
```

Pasta do departamento de pesquisa:

```
FROM WITH "@pesquisas.empresa.com"
```

Pasta do departamento administrativo:

```
FROM WITH "@administrativo.empresa.com"
```

Pasta do departamento de produção:

```
FROM WITH "@producao.empresa.com"
```

Pasta com mensagens urgentes:

```
SUBJECT WITH "URGENTE" || BODY WITH "URGENTE"
```

Pasta do projeto "antares", desenvolvida em conjunto com os funcionários; Pedro, João e Francisco:

```
FROM WITH "pedro" || FROM WITH "joao" ||
FROM WITH "francisco" && SUBJECT WITH "antares" ||
BODY WITH "antares"
```

Pasta contendo os memorandos do primeiro semestre do ano de 1999:

```
BODY WITH "memorando" || SUBJECT WITH "memorando" &&
SENT DATE > 01/01/1999 && SENT DATE < 30/06/1999
```

Pasta contendo os memorandos do segundo semestre do ano de 1999:

```
BODY WITH "memorando" || SUBJECT WITH "memorando" &&
SENT DATE > 01/07/1999 && SENT DATE < 31/12/1999
```

Capítulo 4

Projeto do Protocolo SPOP

4.1. Aspectos de Projeto de um Servidor SPOP

Nesta seção apresentaremos aspectos gerais sobre o projeto de servidores que implementem o protocolo SPOP. O objetivo central desta seção é facilitar o entendimento com relação ao funcionamento interno deste protocolo, bem como facilitar o desenvolvimento de servidores que implementem este protocolo. Relataremos a nossa experiência no desenvolvimento de um servidor, onde abordaremos assuntos como a linguagem de programação, ambiente para desenvolvimento, além de uma visão de mais baixo nível sobre a arquitetura e funcionamento do protocolo SPOP.

A linguagem sugerida para o desenvolvimento do protocolo SPOP é a linguagem Java da *Sun Microsystems* [Sridh97] [Hughe97] [Pew99] [Horst98] [Corne97]. Um dos motivos que nos levaram a escolher Java como a linguagem para o projeto SPOP, está no fato de ser uma linguagem aberta. Uma linguagem é aberta quando permitir que sua funcionalidade original seja facilmente estendida por componentes (*JavaBeans*), por bibliotecas ou "pacotes" de objetos, de acordo com a necessidade do usuário.

Na fase inicial do projeto SPOP era nosso objetivo especificar uma API para o desenvolvimento de aplicações de correio eletrônico, não só aplicações que implementassem o SPOP. Depois do início do projeto (abril de 1997) foi lançada pela *Sun* a biblioteca de classes JavaMail [Sun97], mais precisamente em Dezembro de 1997. A JavaMail surgiu para preencher a necessidade de um ambiente de trabalho (*framework*) para o desenvolvimento de sistemas baseados em correio eletrônico. Esta biblioteca provê um conjunto de classes abstratas, utilizadas para representar todas as entidades que fazem parte do sistema de correio eletrônico. A JavaMail define classes como Mensagem (*Message*), Pasta (*Folder*), Armazenamento (*Store*), Transporte (*Tranport*), etc. Além de toda a sua funcionalidade a JavaMail pode ser usada e estendida para prover suporte a novos protocolos, bem como suportar novas

funcionalidades para o serviço de correio eletrônico, quando necessário. Outro ponto a favor da opção feita pela linguagem Java, foi a existência do pacote JavaServer, que semelhantemente ao JavaMail provê um conjunto de objetos, neste caso especificamente para auxiliar na construção de servidores.

A linguagem Java também possui um conjunto de características desejáveis ao projeto SPOP, tem como independência de plataforma, orientação a objetos, controle automático de lixo (*garbage collection*) e o suporte a várias linhas de execução (*threads*). Estas características são importantes para o desenvolvimento de um servidor eficiente e passível de ser utilizado em diversas arquiteturas.

Foi escolhido o ambiente de desenvolvimento JBuilder 2, da Borland [Armst98] [Jense97]. O JBuilder possui uma série de características interessantes ao desenvolvimento do projeto. O JBuilder possui uma forma simples e direta de pesquisa à documentação das bibliotecas de classes da linguagem Java, através de uma funcionalidade denominada "*Code Browsing Capabilities*". No momento da edição do código o usuário pode utilizar teclas de atalho que requisitem a exibição da parte da documentação em que se encontra a definição da classe que está sob o cursor. Outra funcionalidade associada, é denominada de "*Code Insight*"; durante a digitação do código, quando o usuário completa a digitação do nome de uma classe, automaticamente são listados os métodos da classe em questão, ou quando o usuário está digitando a chamada a um método são mostrados os argumentos necessários para este determinado método.

O JBuilder também possui facilidades para a programação baseada em componentes, tornando fácil a criação destes, além de já possuir várias bibliotecas de componentes previamente instaladas (como AWT, Swing e outras). As bibliotecas de componentes podem ser atualizadas, bem como podem ser adicionadas outras, a critério do usuário. O ambiente do JBuilder também é escalável para aplicações muito grandes e possui um processo de rastreamento de erros.

4.1.1. Aspectos de Implementação

4.1.1.1. Especificação dos SFolders

Nesta seção faremos algumas considerações sobre a implementação dos SFolders, que são as pastas de classificação programada, onde discutiremos algumas sugestões quanto ao seu desenvolvimento. Discutiremos a experiência que tivemos durante o desenvolvimento de um servidor experimental e como implementamos neste servidor a estrutura dos SFolders.

Para representar os SFolders desenvolvemos uma classe denominada "Pasta", contendo basicamente três atributos, quais sejam: o nome da pasta, a sua regra de formação e a lista dos identificadores das mensagens que fazem parte desta pasta. A regra de formação é constituída pelos critérios que as mensagens devem satisfazer para estarem vinculadas a um SFolder específico. Armazenamos a regra de formação em uma estrutura do tipo árvore, na forma de uma hierarquia operadores e critérios, como mostrado na Figura 4.1 abaixo. A estrutura de árvore que suporta a regra estabelece uma precedência entre operações. Com esta estrutura podemos facilmente representar a estrutura de parênteses, suportando desta forma uma maior expressividade na formação de regras.

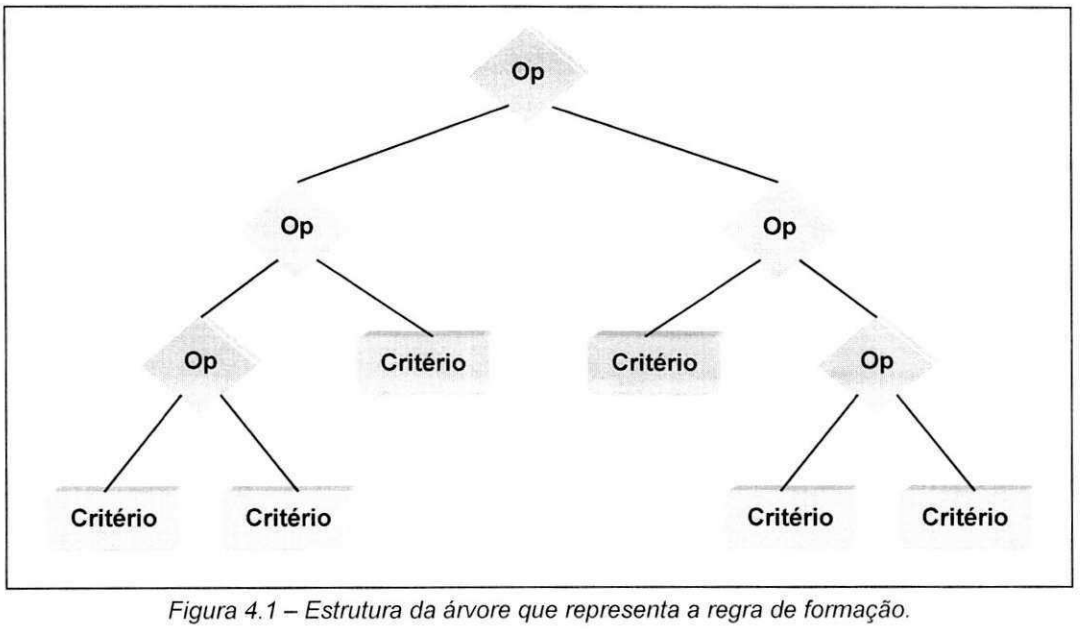

Figura 4.1 – Estrutura da árvore que representa a regra de formação.

A árvore de regra é construído a partir do argumento do comando CREATE correspondente à regra de formação da pasta, o qual é quebrado em tokens e analisado gramaticalmente. Esta análise é realizada através da arvore de gramática mostrada nas Figuras 4.2, 4.3 e 4.4, abaixo.

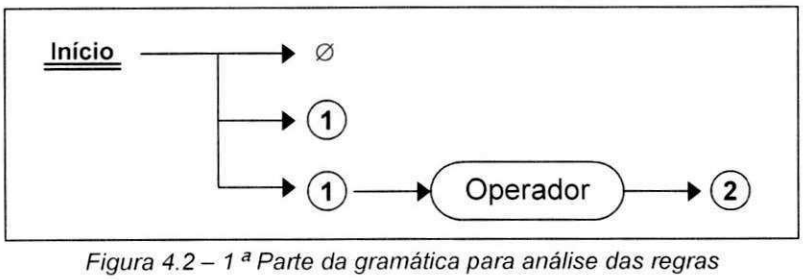

Figura 4.2 – 1ª Parte da gramática para análise das regras

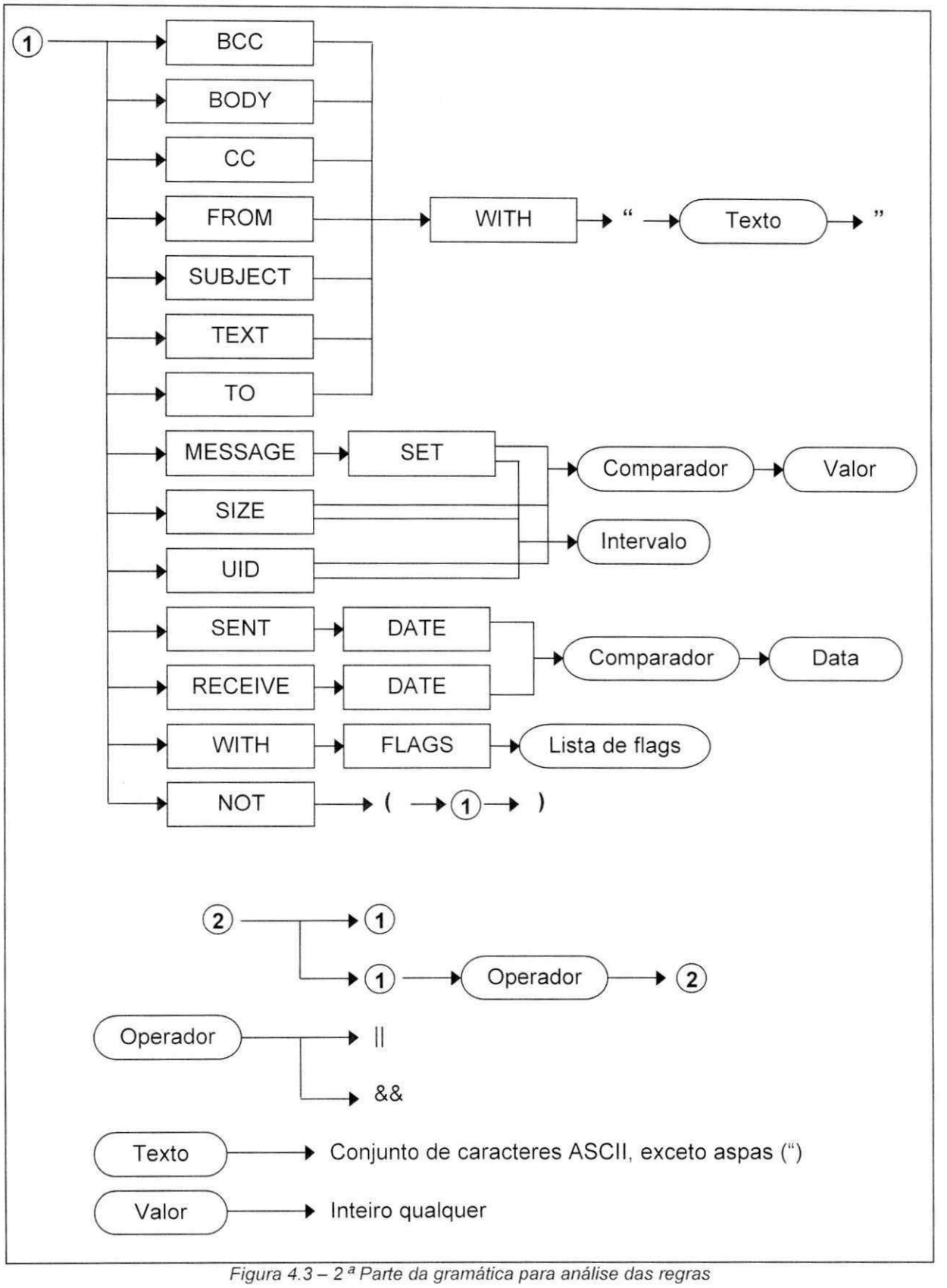

Figura 4.3 – 2ª Parte da gramática para análise das regras

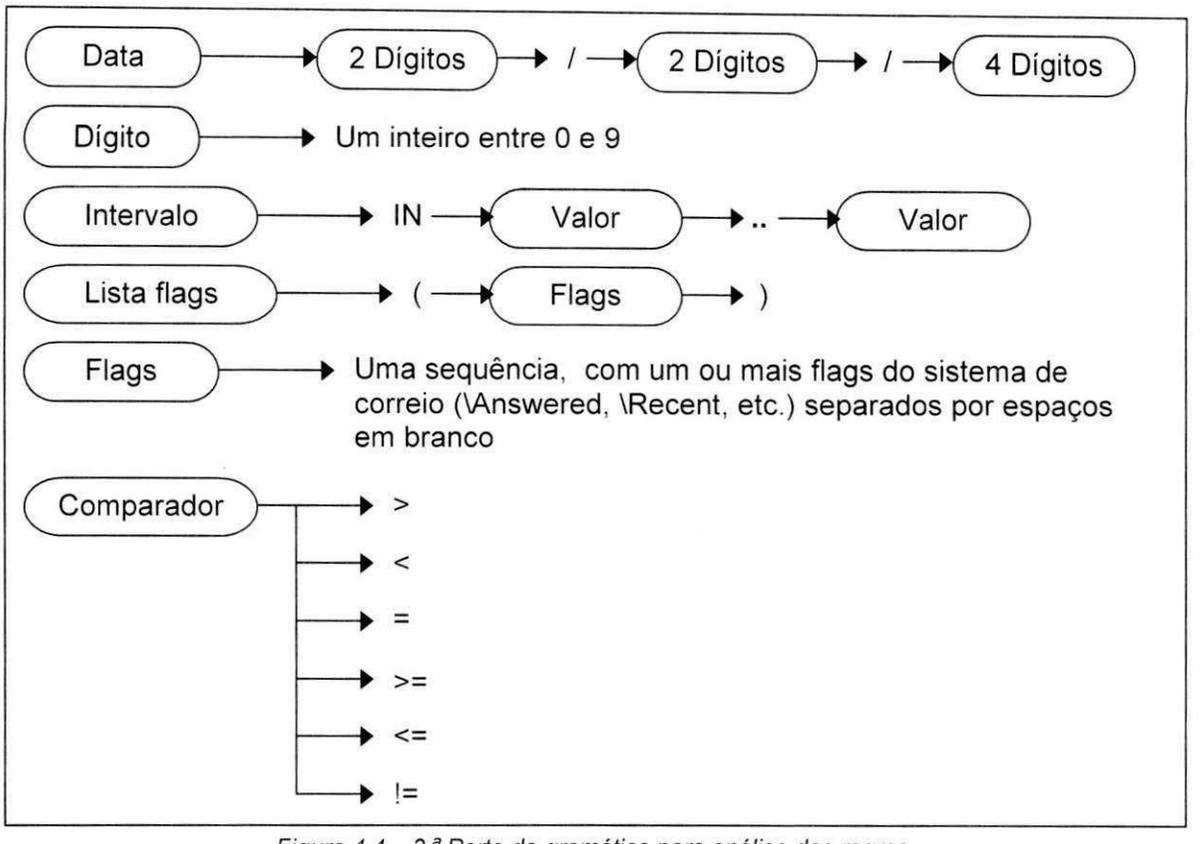

Figura 4.4 – 3ª Parte da gramática para análise das regras.

Depois de identificados, os critérios são inseridos na árvore de regra. Depois de haver sido completamente formada, esta árvore de regra pode ser utilizado para realização de pesquisas na caixa de correio em questão.

O mecanismo de pesquisa de mensagens utilizando esta árvore de regra é realizado utilizando-se quatro estruturas, três registradores e uma pilha. Os três registradores denominam-se "registrador 1", "registrador 2" e "resultado parcial". A pilha denomina-se "parciais". A execução do mecanismo se dá através dos seguintes passos:

1. Efetua-se uma busca na caixa de correio do usuário por mensagens que satisfaçam ao primeiro critério obtido através de um caminhamento *in order* na árvore de regras, o resultado desta busca é um lista com os identificadores universais das mensagens encontradas. Esse resultado é armazenado no "registrador 1";
2. Se o registrador "resultado parcial" estiver vazio, copia-se o conteúdo do "registrador 1" para "resultado parcial";
3. Se o registrador "resultado parcial" não estiver vazio, armazena-se no "registrador 2" o resultado da operação lógica "ou" entre o "resultado

parcial" e "registrador 1". Depois copia-se o conteúdo do "registrador 2" para o registrador "resultado parcial";

4. Se o primeiro operador, também obtido através de um caminhamento *in order*, for um "e", então empilha o conteúdo do registrador "resultado parcial" na pilha "parciais" e prossegue a operação com o registrador "resultado parcial" vazio;
5. Volta ao passo 1 até não existir mais critérios (fim de regra).
6. Se não existirem mais critérios a pesquisar, então desempilha-se um elemento da pilha "parciais" e realiza-se a operação lógica "e" com o registrador "resultado parcial";
7. Volta ao passo 6 até não existir mais elementos na pilha "parciais".

As estruturas "critério" e "operador" guardam as informações necessárias para a efetuação das buscas e completo entendimento da regra como um todo. Os critérios individuais podem ser divididos em quatro tipos fundamentais, pois cada tipo em particular possui um estilo de busca distinto. Os tipos são: texto, flags, data e número. A busca por um critério do tipo texto é basicamente a pesquisa por um determinado trecho de texto em campos do cabeçalho da mensagem (TO, FROM, etc), no corpo (BODY) ou mesmo em todo o código da mensagem (TEXT). A busca por um critério do tipo flag, como o próprio nome já diz é uma pesquisa para indicar a existência ou não de determinados sinalizadores nas mensagens (\Recent, \Deleted, etc). A Busca por critério do tipo data, é uma pesquisa e comparação tomando-se por base determinadas datas de recebimento ou de envio. A busca por critério número é uma pesquisa e comparação tomando-se por base determinados valores inteiros, seja com relação ao tamanho da mensagem ou número de sequência da mesma.

A estrutura onde é armazenada a árvore de regra e pode ser facilmente percorrida pelo servidor para a obtenção do resultado final, que é um lista contento as mensagens resultantes da aplicação da regra. O retorno de cada busca por critério individuais também é uma estrutura vetor, contendo os identificadores das mensagens que satisfizeram o critério individual. As operações lógicas entre critérios são realizadas utilizando-se os vetores resultantes das pesquisas por critério individuais e são semelhantes às operações entre conjuntos.

Na operação lógica "e", utiliza-se um dos dois vetores como base e para cada elemento desse vetor verifica-se a existência de um com o mesmo conteúdo no outro vetor. Se o identificador de uma mensagem estiver nos dois vetores ele é adicionado ao resultado (semelhante à interseção entre conjuntos). Na operação lógica "ou",

utiliza-se um vetor inicialmente como resultado e varre-se o outro vetor para adicionar ao vetor resultado os identificadores que ainda não estiverem no mesmo (semelhante à união entre conjuntos).

O terceiro e último atributo da classe "Pasta" é a lista de mensagens contidas no resultado final da pesquisa que satisfizeram toda a regra de formação da pasta.

Os métodos da classe "Pasta" implementam toda a funcionalidade dos SFolders, como a sua criação, mudança de nome, exclusão, atualização, pesquisa, bem como a recuperação das mensagens que compõem este SFolder, quando o cliente requisitar para examiná-lo.

A criação de um SFolder se dá através dos seguintes passos:

1. A regra de formação é informada pelo usuário como um dos argumentos do comando CREATE, passa por um processo de divisão em critérios individuais e operadores (efetuando-se uma análise gramatical), tendo como produto deste processo uma árvore de regra;
2. A árvore de regra é percorrida e são realizadas as buscas pelos critérios e as operações lógicas entre os resultados da busca pelos critérios, retornando por fim a lista das mensagens que satisfizeram toda a regra de formação;
3. Armazenam-se as informações sobre o SFolder, quais sejam: seu nome, a regra de formação e a lista de mensagens do SFolder;
4. Retorna-se a informação que o SFolder foi criado com sucesso.

Quando um usuário em particular for acessar sua caixa de correio pela primeira vez através de um servidor SPOP é criado automaticamente um SFolder com o atributo especial INBOX, que será a caixa de entrada de mensagens para o usuário. Como a INBOX é a pasta padrão para o armazenamento das mensagens do usuário ela não possui uma regra de formação associada a ela. A INBOX conterá na primeira execução do servidor SPOP todas as mensagens da caixa postal do usuário, pois ainda não existirá nenhum outro *SFolder* criado. Após a criação de outros *SFolders*, permanecerão vinculadas à INBOX apenas as mensagens que não estiverem vinculada a nenhum outro *SFolder*. A pasta INBOX é a única pasta que não pode ser excluída pelo usuário.

Caso o usuário não esteja satisfeito com o nome de um SFolder ele pode ser alterado. A mudança de nome de um SFolder se dá através dos seguintes passos:

1. Identificação da existência do referido SFolder;

2. Se existir, efetua-se a mudança no nome do mesmo;
3. Altera-se as informações de hierarquia dos SFolders que estão em um subnível do SFolder que teve o seu nome alterado.

Quando um SFolder não for mais necessário, ele pode ser excluído. Existem duas formas básicas de se proceder à exclusão de um SFolder: excluindo-se as mensagens que nele estão contidas ou simplesmente apagando apenas o SFolder e recolocando as mensagens na caixa de entrada do usuário (INBOX). A exclusão de um SFolder se dá seguindo os seguintes passos:

1. Identificação da existência do referido SFolder;
2. Se existir, identifica-se se foi informado o argumento NODELMSG, caso esta opção tenha sido selecionada, a lista das mensagens pertencentes ao SFolder são apenas desvinculadas da pasta que vai ser excluída, então voltam a fazer parte da caixa de entrada do usuário;
3. Se não foi selecionada a opção NODELMSG, recupera-se a lista de mensagens pertencentes ao SFolder e tais mensagens são marcadas para serem excluídas, utilizando-se seus identificadores.
4. As informações do SFolder são excluídas;

Caso haja a alteração das informações de alguma mensagem da caixa de correio do usuário sem a interferência do servidor SPOP ou caso haja a chegada de uma nova mensagem, torna-se necessária a atualização dos SFolders do usuário no sentido de manter a sua consistência. A atualização de um SFolder se dá através dos seguintes passos:

1. Identificação da existência do referido SFolder;
2. Recuperação das informações do SFolder;
3. A atualização de um SFolder pode ter duas opções; ou se refaz as buscas em toda a caixa de correio do usuário, ou se faz a busca apenas nas mensagens recém chegadas;
4. É atualizada a lista de mensagens do SFolder e são salvas as suas informações.

Uma vez o SFolder criado, ele está apto a ser utilizado pelos usuários, permitindo o acesso a suas informações, bem como a suas mensagens recuperadas. Para examinar um SFolder são percorridos os seguintes passos:

1. O parâmetro necessário é o nome do SFolder, incluindo toda a sua hierarquia no caso do SFolder não ser uma das subpastas da pasta que está correntemente selecionada;
2. Caso exista, são recuperados os identificadores das mensagens contidas no SFolder;
3. Recuperam-se as mensagens cujos identificadores estão vinculados ao SFolder, podem ainda ser recuperadas partes das mensagens, como por exemplo as informações dos seus cabeçalhos.

Outra importante funcionalidade implementada pelo SPOP são as buscas por critérios específicos. Como já vimos anteriormente estes critérios podem ser de quatro tipos distintos, quais sejam: texto, flag, data e número. Para ilustrar como o servidor realiza estas buscas, veremos os passos seguidos pelo servidor SPOP para processar a busca por um critério do tipo texto:

1. O servidor SPOP tem acesso às mensagens da caixa postal do usuário;
2. De acordo com o campo especificado (BODY, CC, etc.), percorre-se, uma a uma, as mensagens da caixa de correio do usuário e efetua-se a comparação propriamente dita, e em caso positivo coloca-se o identificador universal da referida mensagem na lista de resultados;
3. Retorna-se a lista com os identificadores das mensagens que satisfizeram ao critério.

Para a efetuação das buscas propriamente ditas, temos que a utilização da linguagem Java traz um ganho com relação às outras escolhas, visto a possibilidade de utilização da biblioteca de classes "javax.mail.search". Esta biblioteca possui uma série de subclasses da classe abstrata "SearchTerm" que implementa o método "match". O método "match" retorna valores verdadeiro ou falso indicando se o termo de pesquisa informado foi encontrado em uma mensagem específica. Esta biblioteca oferece métodos de busca já prontos e de uso simples e direto, facilitando o seu uso por parte dos desenvolvedores, como mostrado na Figura 4.5 abaixo:

```
else if (criterio.Campo.equals("body")) {
   BodyTerm padrao = new BodyTerm(criterio.Valor);
   if (padrao.match(mensagem)) {
      String[] id = mensagem.getHeader("Message-ID");
      Resultado.addElement(id[0]);
   }
}
```

Figura 4.5 – Trecho de código, exemplo de utilização da biblioteca javax.mail.search.

No exemplo da utilização da biblioteca javax.mail.search observamos a instanciação da classe "BodyTerm" (subclasse de SearchTerm). Na inicialização da classe "BodyTerm", é informado ao construtor o trecho de texto que se deseja encontrar no corpo da mensagem de cada mensagem. Depois da construção do objeto, a pesquisa na mensagem é feita utilizando-se o método "match", que retorna verdadeiro se o trecho de texto especificado for encontrado no corpo da mensagem. Quando uma mensagem satisfizer ao critério o seu identificador é adicionado ao resultado.

4.2. Considerações Sobre o Desenvolvimento de Clientes SPOP

Sobre o desenvolvimento de clientes que implementem o protocolo SPOP, o ponto chave da discussão é sobre as formas de comunicação entre o cliente e o servidor SPOP. A comunicação entre o cliente e o servidor SPOP pode ser feita de duas formas: utilizando-se sockets ou através de uma API. Quando utiliza-se sockets o desenvolvedor do cliente SPOP tem mais preocupações com relação ao controle e interpretação do diálogo com o servidor. A outra opção é a utilização de uma API para abstrair todo o baixo nível da programação com sockets.

A biblioteca JavaMail, descrita em mais detalhes no Apêndice B deste documento, possibilita a criação e inserção no seu modelo de novas bibliotecas que suportem protocolos novos e específicos, como o SPOP. Existe um documento da Sun Microsystems para auxiliar neste desenvolvimento, denominado "*JavaMail Guide for Service Providers*" [Sun98].

Um dos objetivos do projeto SPOP é o desenvolvimento de uma API que implemente todos os detalhes de uma conexão com o protocolo SPOP, possibilitando a criação rápida e simples de clientes SPOP.

4.3. Visão Geral da Arquitetura

Mostraremos nesta seção a arquitetura que utilizamos, suas principais classes e interfaces. A JavaMail possui elementos que são usados para construir uma interface para o sistema de troca de mensagens. Enquanto esta especificação não define uma implementação em específico, esta API inclui várias classes que implementam a RFC 822 e MIME (protocolos padrão para a troca de mensagens na Internet). Os componentes da arquitetura são distribuídos em camadas, na seguinte disposição:

- A camada de classes abstratas: declara classes, interfaces e métodos abstratos, com a intenção de suportar funções de manipulação de

mensagens eletrônicas suportadas por todos os sistemas de correio. Os elementos desta API podem ser herdados por uma subclasse ou serem estendidos para suportar os tipos padrão de dados, e para fazer interface com os protocolos de acesso a mensagens e transporte, caso necessário;

- A camada de classes de implementação Internet: é responsável pela implementação de parte da camada abstrata, usando os protocolos padrão da Internet - RFC 822 e Mime;
- JavaMail utiliza o JAF para encapsular os dados das mensagens, e manipular os comandos que interagem com os dados. Esta interação com os dados da mensagem deve acontecer através da JAF-aware JavaBeans (os quais não fazem parte efetivamente da arquitetura JavaMail).

Esta especificação da arquitetura em camadas permite aos clientes usar as mesmas chamadas da API JavaMail para enviar, receber e armazenar uma variedade de mensagens usando diferentes tipos de dados, diferentes tipos de armazenamento e usando diferentes protocolos de transporte. As classes da biblioteca JavaMail possuem a seguinte hierarquia:

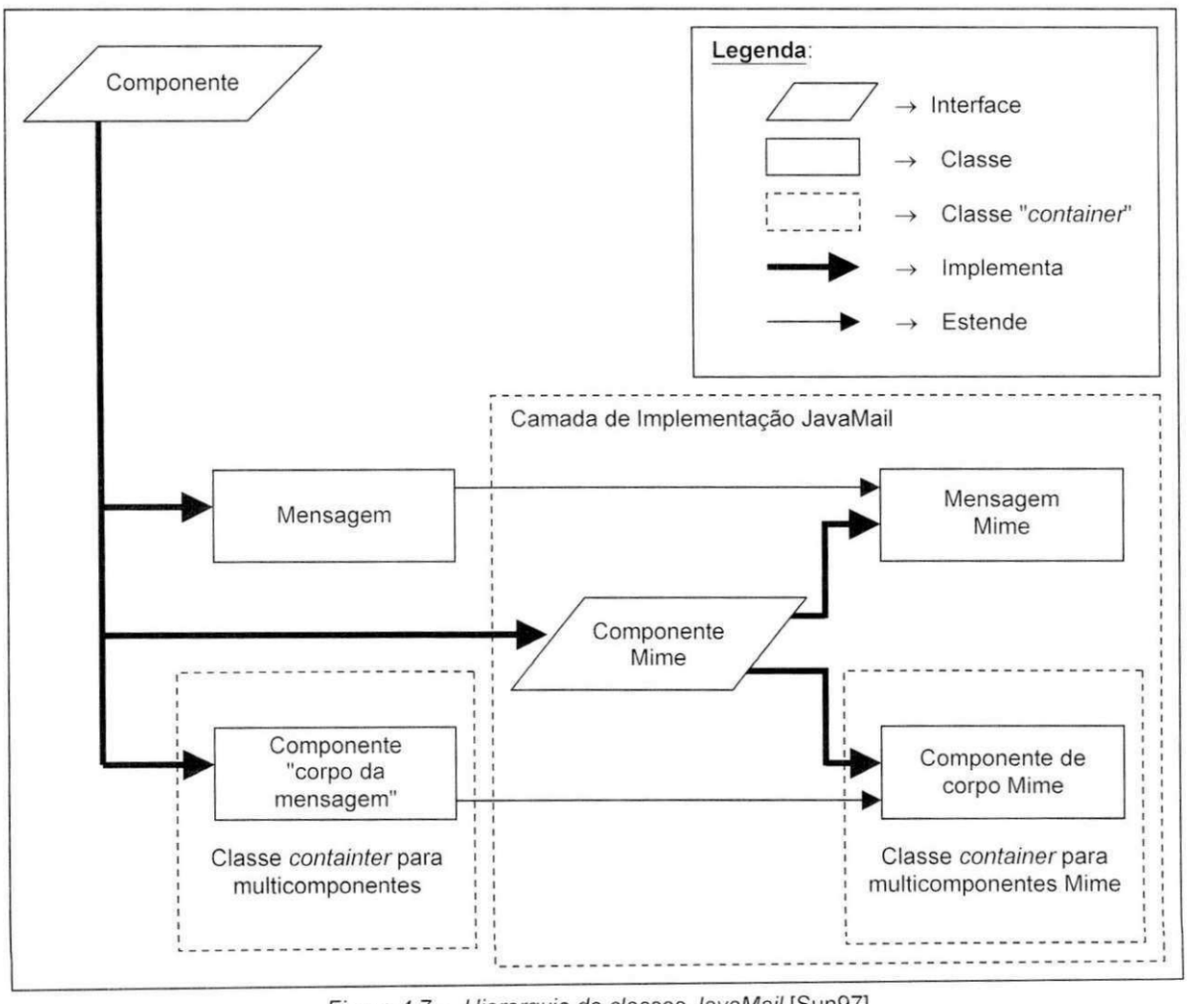

Figura 4.7 – Hierarquia de classes JavaMail [Sun97]

A API JavaMail proporciona o desenvolvimento das seguintes funções, as quais compreendem o processo padrão de manipulação de correio eletrônico:

- Criar uma mensagem de correio, a qual consiste em uma coleção de atributos de cabeçalho e um bloco de dados de algum tipo conhecido, como especificado no campo "tipo de conteúdo" (*Content-Type*) do cabeçalho. A JavaMail utiliza o componente de interface e a classe mensagem para definir uma mensagem de correio. Ele utiliza um objeto manipulador de dados (*DataHandler*) do JAF para conter os dados que irão dentro da mensagem;
- Criar um objeto sessão, o qual autentica o usuário, e controla o acesso ao local onde as mensagens são armazenadas e ao transporte das mesmas;
- Enviar a mensagem para uma lista de recipientes;
- Receber uma mensagem do depósito de mensagens, o qual chamaremos de *mailbox* (caixa de correio);
- Executar um comando de alto nível para recuperar uma mensagem. Comandos de alto nível como *view* e *print*, que são implementados através de componentes JavaBeans JAF.

O framework JavaMail não define mecanismos que suportam a entrega de mensagens, segurança, operações "desconectadas", serviços de diretório ou funcionalidades de "filtro". Segurança, operações desconectadas e suporte a filtragem de mensagens serão adicionados em edições futuras.

Com o desenvolvimento da API do SPOP para ser utilizada com a biblioteca JavaMail teremos um nível de abstração muito confortável para os desenvolvedores de aplicações, como no exemplo das declarações de novos objetos mostrado abaixo.

```
Store mystore = session.getStore(" spop" );
Folder myfolder = mystore.getDefaultFolder();
int TotalMessages = myfolder.getMessageCount();
Message mymessage = myfolder.getMessage(indice);
```

Figura 4.8 – Trecho de código, exemplo do nível de abstração da JavaMail.

Para melhor compreensão de como as classes da biblioteca JavaMail participam dentro do sistema de correio eletrônico, ilustramos na figura abaixo, o processo de manipulação de mensagens na arquitetura JavaMail.

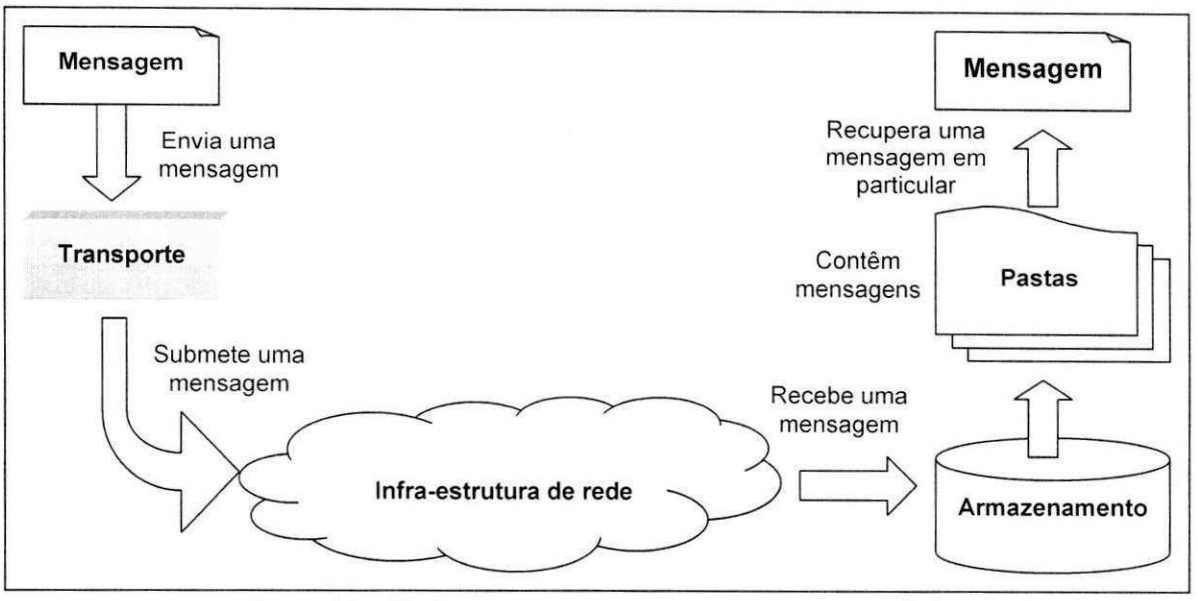

Figura 4.9 – Manipulação de mensagens na Arquitetura JavaMail

A figura acima nos mostra um ilustração clara sobre o papel e o relacionamento entre as classes quem compõem a API JavaMail. A classe mensagem encapsula uma mensagem eletrônica real do sistema de correio eletrônico. É através dela que tanto o programa cliente do usuário de origem, quanto o programa cliente do usuário de destino, efetuam a manipulação da mesma.

A classe Transport encapsula o processo de envio de uma mensagem, onde uma instância desta classe apenas necessita de algumas informações básicas para abstrair completamente ao programa cliente de correio eletrônico os detalhes da transferência de uma mensagem através da infra-estrutura de rede.

A classe Store é utilizada para abstrair o processo de aceso ao MS do sistema de correio eletrônico. É através da classe armazenamento que o usuário é autenticado para o trabalho com uma determinada caixa postal eletrônica. Um dos parâmetros informados para uma instância desta classe é o nome do protocolo que deverá ser utilizado para o acesso e recuperação às mensagens de uma determinada caixa de correio, a API JavaMail provê as opções dos protocolos POP e IMAP. E por fim, a classe pasta é utilizada para simbolizar um conjunto de mensagens contidas em uma pasta de arquivamento do sistema de correio eletrônico.

O projeto SPOP foi desenvolvido para adaptar-se a estrutura especificada pela JavaMail. Dentro do modelo JavaMail, introduzimos os elementos responsáveis pelo comportamento específico do SPOP. O diagrama de classes, mostrando a arquitetura geral do sistema pode ser visto na Figura 4.10 abaixo.

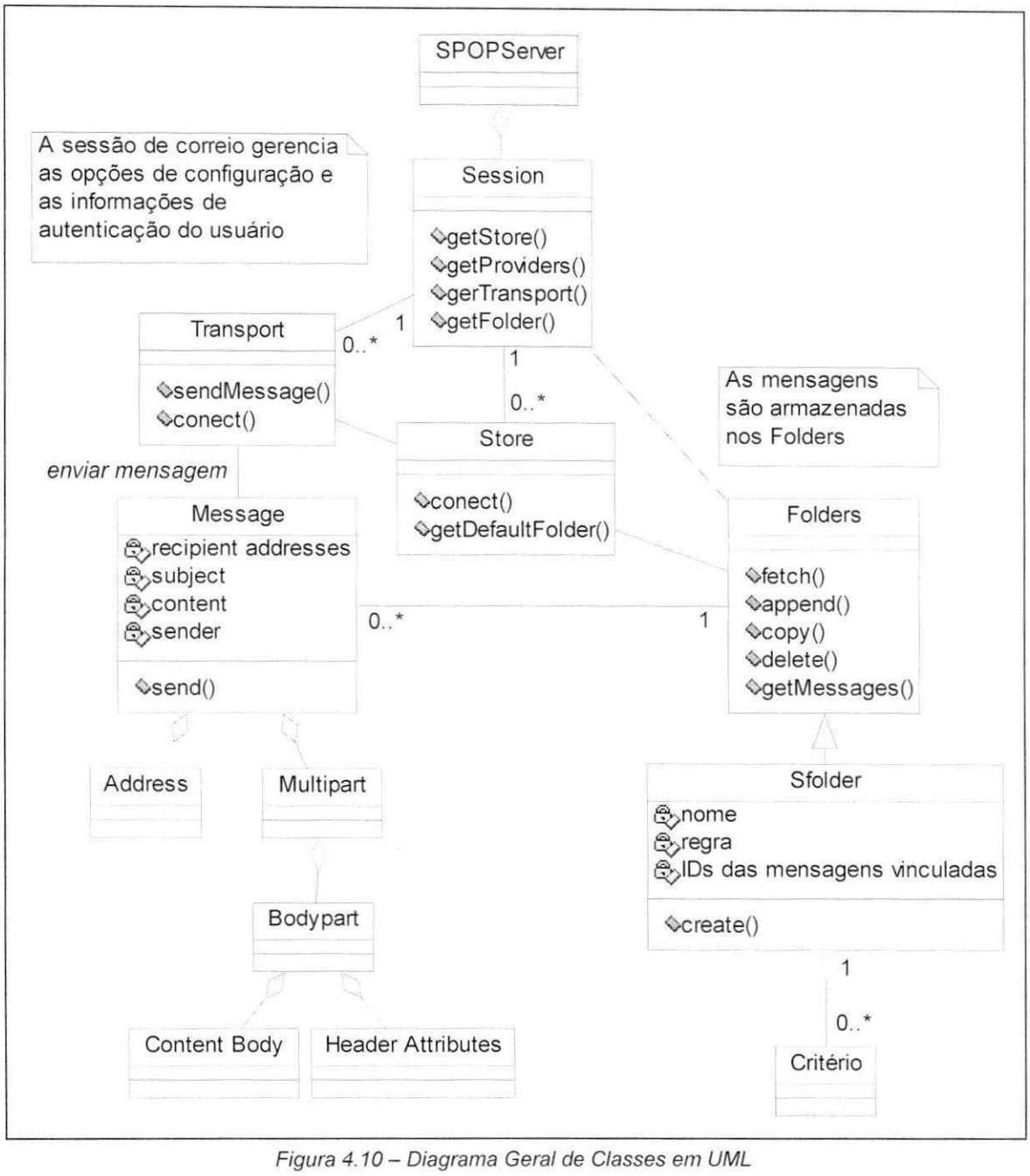

Figura 4.10 – Diagrama Geral de Classes em UML

Para auxiliar na recuperação textual de informação sugerimos uma biblioteca orientada a objetos própria para tal finalidade, como por exemplo a BART (Biblioteca Orientada a Objetos para Apoio a Recuperação Textual) [Bussm95]. A linguagem especificada pela BART auxilia na localização por associação: mesma frase, mesmo parágrafo, etc. Fornece ferramentas para buasca por procimidade: intrevalo de proximidade, proximidade de localização e adjacência. Além de permitir recursos avançados como pesquisa por prefixos e sufixos e utilizando expressões regulares.

4.4. Sumário

O desenvolvimento de um protótipo de um servidor que implementasse o protocolo SPOP seguiu os seguintes passos: 1) foi adotada a metodologia orientada a objetos para desenvolvido o projeto; 2) A estrutura das classes seguiu a especificação da arquitetura JavaMail, apenas foi adicionada uma nova classe para representar os *SFolders*; 3) a linguagem escolhida para o desenvolvimento foi a linguagem Java, devido às inúmeras funcionalidades oferecidas pela linguagem que iam ser úteis durante o desenvolvimento do projeto. Entretanto, a escolha pela linguagem Java se deu antes do lançamento da API JavaMail. Uma vez lançada a API foram feitas algumas adaptações no modelo de classes para que este se moldasse ao esquema de classe especificado pela API JavaMail; 4) foi escolhido o ambiente de programação JBuilder 2.0 da Borland, segundo a familiaridade com os ambientes de desenvolvimento Delphi e C++ Builder, também da Borland; e 5) por fim, a implementação do mesmo.

O servidor protótipo foi desenvolvido *multithreaded*, para possibilitar o trabalho com vários clientes simultaneamente. Para tal foi utilizada a biblioteca de threads da linguagem Java.

Para a implementação do mesmo foi utilizada uma forma semelhante ao POP para tratar do armazenamento das mensagens, o que implica que na prática todas as mensagens continuam armazenadas no servidor em um mesmo local. Toda a divisão das mensagens em *SFolders* acontece de forma lógica, onde as mensagens não são movidas para a área de armazenameto de uma pasta, mas sim vinculadas a esta pasta. Porém este fato é completamente abstraído a partir do programa cliente que utilizará o servidor SPOP.

Foi criada uma classe para representar os *SFolders* no sistema. Não sobrescrevemos a classe "pasta" para possibilitar ao servidor SPOP o acesso às pastas de arquivamento utilizadas pelo IMAP, bem como a pasta padrão utilizada pelo POP. O código fonte da classe que representa os *SFolders* pode ser visto no Apêndice C deste documento. As demais classes permaneceram inalteradas, e foram utilizadas para compor as demais funcionalidades propostas pelo servidor SPOP. O comportamento dos *SFolders* foi modelado segundo os algorítmos vistos na seção 4.1.1. deste documento.

Capítulo 5

Conclusões e Trabalhos Futuros

É inegável, e crescente, importância do correio eletrônico na vida das pessoas, empresas, industrias, instituições, governos ligados de uma forma, ou de outra, à Internet. As tendências indicam que a cada dia cresce o número de pessoas conectadas à Internet. Já estão em discussões projetos que visam popularizar o acesso ao serviço de correio eletrônico, da mesma forma que a população hoje tem acesso ao serviço de correios tradicional. Essas agências de correio do futuro certamente possuirão meios para possibilitar aos seus usuários acessar a sua caixa postal através de um terminal de computador, sem a necessidade de possuir um computador em sua casa, fato este que é um dos limitantes maiores para a popularização do sistema de correio eletrônico a nível mundial.

Protocolos de acesso e recuperação de mensagens têm um papel muito importante dentro da arquitetura do correio eletrônico. Estes protocolos são responsáveis pela entrega das mensagens aos usuários, tarefa sem a qual não teria sentido o serviço de correio eletrônico. Com o aumento no volume de mensagens que trafegam no sistema de correio eletrônico fazem-se necessários novos e mais poderosos protocolos de acesso e recuperação de mensagens. O aumento no volume das mensagens requer que os protocolos de acesso, e recuperação, venham a oferecer meios para que o cliente possa realizar uma seleção das mensagens que devem ser recuperadas em um dado instante. O alvo almejado com o desenvolvimento deste novo protocolo de acesso e recuperação de mensagens, o SPOP, foi desenvolver uma solução bastante flexível e poderosa para a recuperação de mensagens.

As vantagens do SPOP são: ser um protocolo simples, de fácil implementação tanto em clientes quanto em servidores; oferecer a classificação programada de mensagens, onde o cliente informa as pastas em que ele deseja colocar cada tipo de mensagem, para que o mesmo saiba onde encontrar o que deseja a cada momento. Desta forma o SPOP configura-se em um importante aliado do usuário no intento de organizar as suas mensagens de correio eletrônico. Como o seu próprio nome afirma,

o SPOP surge com a função de ser uma agência inteligente de correio eletrônico, auxiliando a figura mais importante da arquitetura do sistema de correio eletrônico que é o usuário final.

Durante o desenvolvimento do SPOP encontramos alguns problemas. Um deles foi distância lógica entre o mapeamento de um desejo do usuário e a classificação de um certo tipo de mensagem numa expressão, utilizando uma determinada linguagem que expresse este desejo da melhor forma possível. Para enfrentar este problema optamos por desenvolver uma linguagem simples, com grande poder de representação.

Gostaríamos de registar aqui também algumas das dificuldades que tivemos com relação ao desenvolvimento do projeto. Em primeiro lugar, uma consideração a respeito da área de correio eletrônico que é atualmente uma área bastante ativa. Como exemplo deste fato, tivemos o lançamento da API JavaMail durante o desenvolvimento do projeto. Em segundo lugar, selecionamos inicialmente um escopo muito grande para o projeto, que envolvia a idealização e desenvolvimento do protocolo, o desenvolvimento de um servidor completo e um cliente com todas as funcionalidades, o que nos fez avançar lentamente e mudar de direção algumas vezes durante o desenvolvimento do projeto.

Com relação ao desenvolvimento do projeto, estamos satisfeitos com este primeiro passo, mesmo sendo um passo menor do que estava previsto para ser trilhado. Este projeto ganhou mais importância quando descortinou um horizonte de possibilidades na área, e mostrou quanto ainda resta por ser feito, movidos pela mesma motivação inicial para este projeto. Não poderíamos ser definitivos em nossa solução, visto que esta esbarrou na forma, ou ausência de formatação específica para o armazenamento de mensagens de correio eletrônico. A ausência de mecanismos intrínsecos ao armazenamento de mensagens, como por exemplo a tecnologia de banco de dados textual, impedem que estas mensagens sejam recuperadas de forma otimizada.

A proposta de integrar aspectos de banco de dados textual na forma de armazenamento das mensagens de correio eletrônico, atenderia a principal deficiência do protocolo SPOP que á a dissociação entre as mensagens e suas informações de contexto. Na estrutura do SPOP as mensagens continuam a ser armazenadas em um MS tradicional e as informações de contexto, como as informações sobre os SFolders, são armazenadas em uma outra estrutura. Com a dissociação entre essas

informações, o maior problema está na manutenção da consistência dos SFolders, que passa a ser a maior preocupação durante a execução de um servidor SPOP.

Em uma nova forma de armazenamento, utilizando-se um MS com as características de um banco de dados textual, a caixa de correio do usuário poderia constituída por um conjunto de duas tabelas protegidas e só acessíveis pelo dono da caixa de correio após prévia autenticação. Uma das tabelas conteria todas as mensagens da caixa de correio do usuário, a outra conteria as informações sobre os SFolders, como os seus nome e regras de formação. Na tabela das mensagens do usuário, cada registro corresponderia às informações de uma mensagem específica, com atributos como as suas informações de cabeçalho (TO, FROM, CC, etc.), o corpo da mensagem, os flags associados à mesma, além dos nomes dos SFolders que contêm as mensagens. Nessa arquitetura, a consistência seria atingida automaticamente com a utilização de gatilhos (*Triggers*). Qualquer mensagem que for inserida na tabela de mensagens, ou qualquer mensagem que sem a intervenção do SPOP tiver seus atributos alterados, dispara um gatilho que é responsável pela manutenção da consistência. O disparo de um gatilho daria início a uma pesquisa para identificar se a mensagem que deu início ao processo pertence a algum SFolder, isto é, satisfaz a regra de formação deste SFolder.

A regra de formação para um SFolder seria uma construção direta em uma linguagem de consulta a banco de dados textual. A pesquisa na caixa de correio do usuário por mensagens que satisfaçam uma determinada regra seria a aplicação de um comando de busca desta linguagem.

Essa nova estrutura de armazenamento traria aos usuários a garantia de um acesso (e recuperação) rápido e flexível às informações contidas na sua caixa de correio. Além disso as tabelas representando a caixa de correio do usuário poderiam alimentar outras tabelas, ou utilizar outras tabelas para retornar ao usuário uma informação mais completa e consistente de uma maneira rápida e fácil. Como estamos tratando de mensagens de correio eletrônico como parte de um banco de dados textual maior, poderíamos utilizar o próprio ambiente do correio eletrônico para recuperar informações contidas neste banco de dados, por exemplo: em uma empresa o gerente de desenvolvimento poderia recuperar junto com uma mensagem de um dos seus subordinados, informações sobre o funcionário e os projetos em que está envolvido.

Esperamos que a abordagem utilizada no SPOP levante a discussão sobre como são armazenadas as mensagens de correio eletrônico, além de um

questionamento se estas formas estão atendendo às necessidades dos usuários. A tendência que está se observando na Internet é o aumento no volume de mensagens de correio eletrônico, o questionamento maior neste caso é se a atual forma de armazenamento e recuperação de mensagens está pronta para atender todas as necessidades dos usuários.

O projeto SPOP deixa como contribuição maior a estrutura dos SFolders, que são as pastas de classificação programada, a SFDL, que é a linguagem de construção de regras de formação, além de levantar aspectos importantes quanto à atual estrutura do acesso e recuperação de mensagens de correio eletrônico. Os SFolders representam uma nova forma de trabalhar com as mensagens de correio eletrônico, uma forma de auxiliar o usuário no seu trabalho de administração da sua caixa de correio, através de uma ferramenta simples e poderosa. A SFDL pode muito bem ser utilizada fora do contexto do SPOP, como um meio simples e direto para a estipulação de critérios para a classificação de mensagens de correio eletrônico. Finalmente, uma outra contribuição do SPOP está na esfera teórica, e diz respeito às críticas e sugestões à atual estrutura do acesso e recuperação de mensagens que podem ser utilizados como base para trabalhos futuros nesta área.

Comparando as funcionalidades do SPOP com as soluções do POP e do IMAP, temos que o SPOP possui todas as funcionalidades do POP, além de oferecer uma abordagem distinta do IMAP para uma mesma tarefa, que é a recuperação eficiente de mensagens.

Na Tabela 5.1, temos uma série de comparações entre o SPOP, POP, IMAP e o serviço de correio eletrônico do Lotus Notes. Foram identificadas algumas áreas em que há distinções claras entre uma abordagem e outra. A primeira distinção está relacionada à forma do armazenamento das mensagens em pastas, onde é mostrada como cada abordagem trabalha com a estrutura de pastas. As únicas duas abordagens que trabalham com pastas de arquivamento são o IMAP e o SPOP, com a diferença que o SPOP adiciona o conceito de pastas com classificação programada. O Notes não trabalha com pastas, mas é o único que trabalha com funcionalidade de visões de uma caixa de correio (como base de dados) e possibilita a categorização das mensagens.

A segunda distinção é o nível de complexidade de utilização, que estima o grau de dificuldade para o desenvolvimento de um cliente que implemente o protocolo. Dentre as três abordagens comparadas a solução mais simples é o POP, seguida do SPOP e do Notes, e por fim, o IMAP.

A terceira distinção trata da compatibilidade com as outras soluções, que nos informa quais os usuários que utilizam uma das opções, pode utilizar uma das outras sem prejuízo. O POP reconhece apenas as mensagens contidas na INBOX, como a outra opção que também mantêm as mensagens na INBOX é o SPOP, o SPOP pode ser utilizado sem prejuízos por um usuário do POP. O IMAP reconhece as mensagens da INBOX e acrescenta as pastas de arquivamento próprias do IMAP, como a outra opção que também reconhece as pastas do IMAP é o SPOP, o SPOP pode ser utilizado sem prejuízos por um usuário do IMAP. O SPOP reconhece as mensagens da INBOX, as pastas do IMAP e acrescenta os *SFolders*. O Notes só reconhece a sua própria estrutura de armazenamento, portanto não é compatível com nenhuma das outras soluções.

A quarta distinção trata da estrutura do MS, que é a forma pela qual as mensagens são armazenadas enquanto não são recuperadas pelos clientes. O POP armazena no MS apenas as mensagens. O IMAP armazena além das mensagens informações adicionais sobre as mesas (como os flags) e sobre as pastas. Com o SPOP são armazenadas apenas as mensagens no MS, as informações adicionais são armazenadas em estruturas auxiliares no próprio SPOP. O Notes não utiliza o MS.

A quinta distinção diz respeito à forma de acesso e recuperação das mensagens. A arquitetura do POP prevê que um usuário acesse as suas mensagens sempre a partir de um mesmo local. Já o IMAP e o SPOP prevêem que as caixas postais eletrônicas dos usuários sejam acessadas de qualquer ponto da rede, tendo a mesma visão da referida caixa postal eletrônica independentemente do local de onde está acessando, o que chamamos de acesso distribuído. O Notes permite que o usuário possa ter acesso a sua base de dados de mensagens a partir de qualquer cliente Notes distribuído.

A sexta distinção está nas formas utilizadas para prover seletividade nas recuperações das mensagens. As únicas duas abordagens que oferecem seletividade na recuperação de mensagens são o IMAP e o SPOP. O IMAP através do comando SEARCH que retorna os identificadores das mensagens com um determinado critério, e o SPOP na forma de pastas especiais, os *SFolders*, que têm vinculada a si uma lista das mensagens que possuem determinados critérios. O Notes possui várias funcionalidades de auxílio a recuperação seletiva, são elas: o comando FIND que realiza buscas textuais na base de dados de mensagens do usuário, visões da caixa de correio e exame de elementos eu foram divididos anteriormente em categorias.

A sétima distinção está nas formas utilizadas para prover um melhor desempenho na recuperação das mensagens. As únicas abordagens a oferecer formas de tornar mais eficientes a conexão entre o servidor e o cliente são as abordagens do IMAP e do SPOP: ambas as soluções trabalham com uma estrutura de múltiplos comandos. O trabalho com múltiplos comandos só é possível através da utilização de rótulos em cada comando, indicando que a resposta específica para um comando tem de ter o mesmo rótulo utilizado para enviar o comando.

A oitava distinção está nas vantagens que a solução oferece para que seja realizada a administração automática da caixa de correio do cliente. A única abordagem a oferecer tal facilidade é o SPOP, através das pastas de classificação programada.

Funcionalidade	POP	IMAP	Lotus Notes	SPOP
Pastas	Não trabalha com pastas, possui apenas a INBOX	Trabalha com a INBOX, pastas hierárquicas e pastas especiais	Pode categorizar documentos e pode efetuar visões da caixa de correio	Trabalha com as pastas suportadas pelo IMAP e adiciona os SFolders
Nível de complexidade na utilização	Simples	Complexo	Intermediária	Intermediária
Compatibilidade	Compatível com o SPOP	Nenhuma	Nenhuma	Compatível com o POP e o IMAP
Estrutura do MS	Não armazena nenhuma informação adicional sobre a mensagem no MS	Guarda as informações sobre o estado das mensagens e sobre as pastas	Não utiliza o MS, arquiva as mensagens na forma de base de dados	Também não armazena no MS nenhuma informação adicional sobre a mensagem. É o servidor SPOP quem guarda as informações necessárias
Forma de acesso	Centralizado em uma única máquina cliente	Acesso distribuído	Acesso distribuído	Acesso distribuído
Recuperação seletiva	Nenhuma	Pesquisa remota com o comando SEARCH e a possibilidade de recuperação apenas de partes das mensagens	Pesquisa com o comando FIND, visões e exame de categorias	Pastas com classificação programada e a possibilidade de recuperação apenas de partes das mensagens
Aceleração de desempenho	Nenhuma	Múltiplos comandos rotulados (*pipeline*)	Nenhuma	Múltiplos comandos rotulados (*pipeline*)
Administração automática	Nenhuma	Nenhuma	Nenhuma	Pastas de classificação programada
Forma de Interação	Troca de comandos	Troca de comandos, mais cache local independente	Troca de comandos de baixo nível que são abstraídos pelos clientes Lotus Notes	Troca de comandos, mais filas de tarefas a serem realizadas no servidor e cliente

Tabela 5.1 – Comparativo entre as funcionalidades do POP, IMAP, SPOP e Lotus Notes

A nona e última distinção está na forma de interação entre um cliente e um servidor que implementam este protocolo. A forma básica de iteração entre cliente e

servidor é a troca de comandos, adotada por todas as soluções. O IMAP acrescenta o conceito de cache local independente, que permite ao servidor IMAP enviar dados para o cliente IMAP, sem que seja necessário este os solicitar explicitamente. O SPOP, além da troca de comandos, acrescenta no servidor uma fila de respostas pendentes e no cliente uma fila de comandos pendentes, criando um esquema de *buffers* para agilizar a comunicação. No Notes temos que a comunicação entre o cliente e servidor se dá através da troca de instruções de baixo nível, mas o usuário tem abstraído todo este processo através dos clientes do Notes. Os clientes do Notes possuem interface gráfica, que faz com que os usuários não precisem saber como de fato está sendo feita esta comunicação.

Podemos dividir os trabalhos futuros em duas classes. A primeira classe de trabalhos futuros é composta pelos trabalhos que já estão em desenvolvimento, com vista ao encerramento da primeira fase do projeto SPOP. A segunda classe de trabalhos futuros é composta por trabalhos sugeridos e que poderão fazer parte de uma segunda fase do projeto SPOP.

Dentre os trabalhos da primeira classe, temos o desenvolvimento do *Internet Draft* que definirá o protocolo SPOP e o tornará público e aberto a críticas. Estamos em fase de desenvolvimento da API SPOP para a utilização em conjunto com a API JavaMail. Esta API tornará a construção de clientes SPOP mais simples e cada vez mais poderosos, fato que corroborará para a divulgação do SPOP. Ainda na primeira classe de trabalhos futuros, temos a disponibilização de uma agência de correio eletrônico Internet completa, contendo um servidor SMTP, um servidor POP3 e um servidor SPOP, prontos para utilização, também objetivando a divulgação do protocolo.

Sugerimos como parte da segunda classe de trabalhos futuros o desenvolvimento da RFC SPOP, efetuando-se anteriormente uma análise de todas as críticas geradas pela divulgação do protocolo SPOP por meio da sua *Internet Draft*, e realizando-se os ajustes necessários. Uma vez lançada a RFC, sugerimos que a API SPOP seja atualizada para comportar os ajustes no protocolo. Sugerimos adicionalmente o desenvolvimento de um cliente SPOP experimental de código aberto, que seria disponibilizado na Internet, para que usuários possam utilizá-lo e comprovar as suas vantagens.

Uma vez levantada a discussão, sugerimos a proposta formal de uma nova arquitetura de correio eletrônico, que venha a propor uma extensão do SMTP para utilizar uma estrutura de banco de dados textual na forma de armazenamento de

mensagens. Decorrentemente sugerimos uma nova proposta para o protocolo SPOP, o que chamaríamos de SPOP-SRIT (Sistema de Recuperação de Informação Textual), onde o protocolo SPOP faria a recuperação das mensagens através da estrutura de banco de dados. Tratando o sistema de correio eletrônico como um grande banco de dados o protocolo de acesso e recuperação de mensagens de correio eletrônico têm a possibilidade de atender melhor as requisições dos usuários.

Apêndice A

Envio e Recebimento de Mensagens no Correio Eletrônico Internet

A maior funcionalidade do sistema de correio eletrônico Internet está concentrada em enviar e receber mensagens. Para a melhor compreensão de como se dão estes processos no correio eletrônico Internet, serão mostrados, como exemplo, dois cenários envolvendo o envio e a recepção de mensagens eletrônicas na Internet utilizando o protocolo POP.

No primeiro cenário, ambos os usuários pertencem ao mesmo domínio TCP/IP ("guys.com") e a mesma agência de correio eletrônico, que contêm um servidor SMTP (SMTPD) e um servidor POP (POPD). Temos dois usuários com os nomes deAlbert e Barney, os quais possuem UAs que têm acesso ao correio eletrônico Internet (Eudora e Outlook Express) nos seus computadores, UAs configurados para acessar os servidores SMTP e POP em questão. Cada computador está executando um ambiente operacional diferente, o servidor executa o UNIX, o computador de Barney executa o *Windows NT Workstation*, e o computador de Albert executa o *Windows 95*, como mostrado na Figura 2 [Hughe98]. No exemplo, o usuário Albert deseja enviar uma mensagem eletrônica ao usuário Barney.

O rótulo S1 na Figura 2 indica o primeiro passo no processo de envio da mensagem, no qual o usuário Albert inicia a execução do seu UA e compõe um nova mensagem endereçada para Barney (Barney@guys.com). Então com a mensagem pronta, ele dá início ao processo de envio ativando a parte correspondente da interface do UA.

O rótulo S2 indica o segundo passo no processo de envio da mensagem, no qual o UA em resposta ao botão clicado pelo usuário efetua uma conexão de rede com o servidor na porta 25, a qual é utilizada pelo servidor SMTP, que aceita a conexão.

No próximo passo, indicado pelo rótulo S3, com a conexão estabelecida com o servidor SMTP, o UA do usuário Albert identifica o mesmo como remetente e o usuário Barney como destinatário, os quais são ambos aceitos pelo servidor SMTP. Então o UA transfere a mensagem ao servidor SMTP. De posse da mensagem o servidor SMTP observa as informações sobre o endereçamento da mensagem, e identifica que

esta é destinada a um usuário local. Então, repassa a mensagem para o MS arquivar na caixa de correio do respectivo destinatário. Uma vez completado o envio da mensagem o UA do usuário Albert encerra a conexão com o servidor.

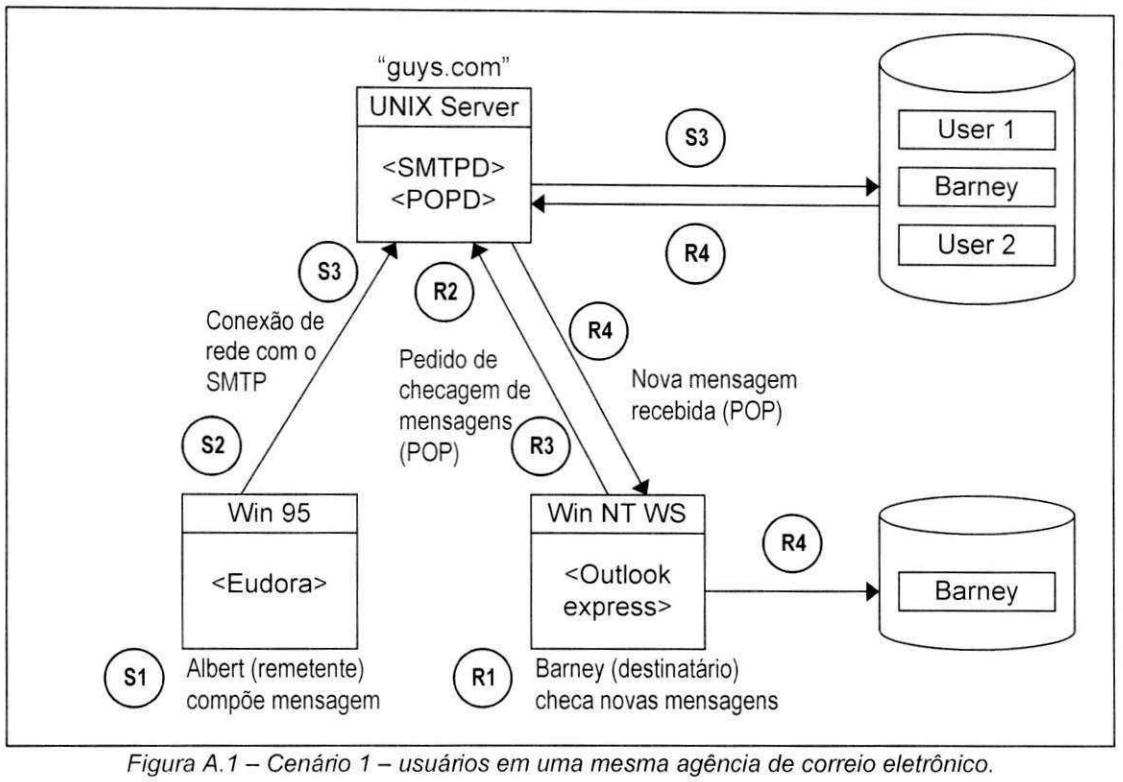

Figura A.1 – Cenário 1 – usuários em uma mesma agência de correio eletrônico.

Para completar a troca da mensagem eletrônica, em algum momento, o usuário Barney tem que recuperá-la do servidor. O rótulo R1 da Figura 2, indica o primeiro passo do processo de recuperação da mensagem, no qual o usuário Barney inicia o seu UA e clica no botão correspondente a verificação de novas mensagens na interface do UA.

No rótulo R2 temos a indicação que o UA do usuário Barney, em resposta a ativação da interface, efetua uma conexão de rede com o servidor na porta 110, que é a porta usada pelo servidor POP, que aceita a conexão.

No rótulo R3 temos a indicação que uma vez a conexão estabelecida com o servidor POP, o UA do usuário Barney identifica o mesmo como um usuário válido no sistema e informa a sua senha de acesso corretamente, a qual é aceita, autenticando assim o usuário.

O Rótulo R4 indica que o UA do usuário Barney fez um pedido através do servidor POP para recuperar a lista das mensagens que estão na sua caixa de correio armazenada no MS. Ao recuperar a lista das mensagens, o UA identifica que existe uma nova mensagem. Então o UA recupera a mensagem por completo através da conexão de rede e armazena no MS local. O usuário Barney não selecionou no seu

UA a opção para deixar as mensagens no servidor, então o UA informa ao servidor POP para apagar a mensagem que foi recuperada. Uma vez o UA identificando que não existe novas mensagens, encerra a conexão de rede com o servidor. Então os dados da mensagem são mostrados na interface do UA do usuário Barney, o qual pode ler a mensagem, respondê-la, imprimi-la, repassá-la para um outro usuário, apagá-la, além de outras formas de manipulação das suas mensagens permitidas pelo UA.

No segundo cenário exemplo, temos dois usuários em duas agências de correio diferentes. Temos dois domínios TCP/IP distribuídos na Internet, cada um com uma agência de correio Internet (servidores SMTP e POP). O primeiro domínio já foi mostrado no exemplo anterior, e tem o nome de "guys.com". O segundo domínio tem o nome de "gals.com", como pode ser visto na Figura 3. Dois usuários serão utilizados neste exemplo, são eles o usuário Albert que pertence ao domínio "guys.com" e a usuária Alice que pertence ao domínio "gals.com", ambos possuem UAs em seus computadores, configurados para os respectivos servidores SMTP e POP. Nesse nosso segundo cenário exemplo o usuário Albert deseja enviar uma mensagem eletrônica para Alice.

O rótulo S1 na Figura 3 indica o primeiro passo no processo de envio da mensagem, onde o usuário Albert, semelhantemente ao primeiro cenário, inicia o seu UA e compõem um nova mensagem endereçada para "Alice@gals.com". Então com a mensagem pronta, ele dá início ao processo de envio clicando no botão correspondente da interface do UA.

O rótulo S2 indica o segundo passo no processo de envio da mensagem, no qual em resposta à ativação da sua interface, o UA efetua uma conexão de rede com o servidor "guys.com" na porta 25, a qual é a porta usada pelo servidor SMTP, que aceita a conexão.

No terceiro passo no processo, indicado pelo rótulo S3, o UA do usuário Albert identifica o mesmo como remetente e a usuária Alice, no servidor "gals.com", como a destinatária, e UA envia a mensagem. O servidor SMTP reconhece que a mensagem é destinada a alguém em outro domínio, então ele não armazena em nenhuma caixa de correio do MS central. O SMTP efetua um consulta ao DNS para identificar o local mais indicado para repassar a mensagem, objetivando que a mensagem chegue o seu destino. Eventualmente, a resposta obtida será que o servidor responsável por mensagens para o referido domínio é o "smtp-mail.gals.com".

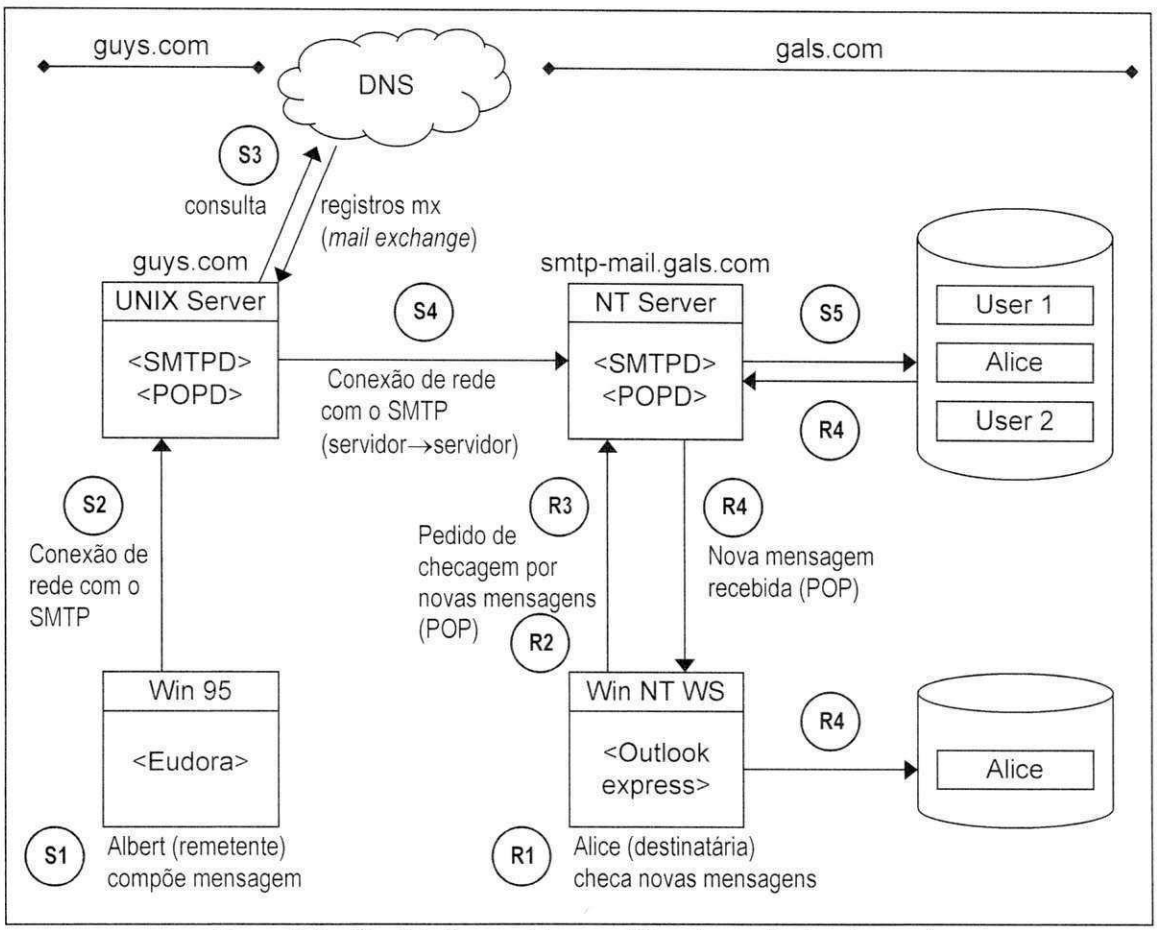

Figura A.2 – Cenário 2 – usuários em diferentes agências de correio

O rótulo S4 indica o quarto passo no processo de envio, nesse ponto com a informação do servidor encarregado de receber mensagens eletrônicas para o domínio "gals.com", o SMTP do domínio "guys.com" efetua uma conexão de rede com o servidor SMTP da máquina "smtp-mail.gals.com". Ao servidor de destino a conexão se assemelha a uma conexão de um cliente local e é manipulada da mesma forma.

No quinto passo no processo de envio, indicado pelo rótulo S5, o SMTP de destino identifica o usuário Albert do domínio "guys.com" como o remetente e a usuária Alice do domínio "gals.com" como a destinatária, ambos são aceitos pelo servidor SMTP. O SMTP de origem envia a mensagem. O SMTP de destino identifica que a mensagem é endereçada a alguém cadastrado localmente, e armazena na caixa de correio correspondente do MS. Após a transferência, o servidor de origem encerra a conexão.

Para completar o processo de transferência da mensagem, depois de efetivado o envio a usuária Alice deve recuperar a mensagem. Para tal, ela inicia o seu UA, como indicado pelo rótulo R1 na Figura 3, e clica no botão da interface do UA que dá inicio ao processo de verificação de novas mensagens. O rótulo R2 indica que o segundo passo no processo de recuperação da mensagem, no qual o UA da usuária

Alice efetua uma conexão com o servidor de correio local na porta 110, que aceita a conexão. O rótulo R3 indica o terceiro passo no processo de recuperação da mensagem, no qual o UA da usuária Alice identifica a mesma como uma usuária cadastrada no serviço POP e informa a sua senha de acesso. A senha de acesso informada está correta e a autenticação da usuária é bem sucedida.

No quarto passo do processo de recuperação da mensagem, indicado pelo rótulo R4, temos que o UA da usuária Alice requisita a lista das mensagens que estão na sua caixa de correio do MS, identificando após a resposta do servidor POP que existe uma nova mensagem. Recupera a mensagem completamente através de uma conexão de rede e armazena no MS local. Diferentemente do usuário Barney, a usuária Alice escolheu no seu UA a opção de deixar a mensagem no servidor, então, seu UA não requisita ao servidor POP que a mensagem, após a transferência, deva ser removida. O UA da usuária Alice toma nota do identificador único da mensagem (UIDL) para não efetuar a transferência desta mensagem novamente.

O UA da usuária Alice identifica que não existe nenhuma nova mensagem, então envia o comando requisitando o fim da conexão. Em seguida, os dados da mensagem são mostrados na interface do UA da usuária Alice, mostrando que foi enviada pelo usuário Albert do domínio "guys.com", assim a usuária Alice poderá ler a mensagem, respondê-la, imprimi-la, repassá-la para um outro usuário, apagá-la, além de outras formas de manipulação permitidas pelo seu UA. Ainda com relação a arquitetura do sistema de correio eletrônico Internet, é possível informar a um MTA do correio eletrônico Internet para sempre enviar as mensagens através de um mesmo servidor de correio, ao invés de fazer a distribuição diretamente às máquinas destino. Basicamente, todos os outros tipos de transações envolvendo os servidores de correio eletrônico acabam recaindo em um destes dois cenários. Toda a funcionalidade adicional, como responder, encaminhar uma mensagem, e etc, são efetuadas pelo UA.

Apêndice B

Biblioteca de Classes *JavaMail*

Em dois anos, desde o primeiro *release* da linguagem, Java tem se desenvolvido para tornar-se uma plataforma completa. Java atualmente provê todas as funcionalidades necessárias para a implementação de um sistema operacional completo, além de prover importantes recursos como: computação distribuída, com RMI e pontes para a arquitetura CORBA, arquitetura de componentes (*JavaBeans*), camada *middleware* (*Enterprise Java Beans*), conjunto de ferramentas para a construção de servidores (*server toolkit*), e o ambiente *WebTop* para o desenvolvimento de aplicações voltadas para a WWW. Da mesma forma que as funcionalidades da linguagem, as aplicações Java têm provado o sucesso da linguagem e têm se desenvolvido com a linguagem. Uma das necessidades atuais é um *framework* baseado na linguagem Java para o trabalho com correio eletrônico. A API JavaMail descreve uma especificação que satisfaz esta necessidade.

A JavaMail provê um conjunto de classes abstratas que definem os objetos de um sistema de correio eletrônico, são elas: *Message* (Mensagem), *Store* (Armazenamento), *Transport* (Transporte). A JavaMail possui também subclasses concretas das classes abstratas, são elas: *MimeMessage* (Mensagem no padrão Mime), *MimeBodyPart* (Componente do corpo de uma mensagem Mime).

O público alvo para esta tecnologia pode ser dividido em três classes. A primeira classe é composta pelos desenvolvedores de softwares clientes, servidores e middleware, interessados em aplicações baseadas em correio eletrônico, em Java. Na segunda classe temos os desenvolvedores de aplicações que necessitem permitir o trabalho com correio eletrônico. E por fim, na terceira classe temos os provedores de serviços que necessitem implementar formas específicas de acesso e transferência de mensagens (como exemplo, um serviço de PAGERS onde as mensagens são enviadas para estes via correio eletrônico).

Como metas e princípios da especificação desta tecnologia, temos que a API JavaMail foi desenvolvida para satisfazer os seguintes requisitos (de desenvolvimento e execução):

- Ser simples, e ter uma especificação de classe fácil para o desenvolvedor aprender a implementar;
- Usar conceitos familiares e modelos de programação que suportam o desenvolvimento de código que tem boa interface com outras APIs. Utilizando tratamento de exceções já conhecido (*exception handling*), da mesma forma que manipulação de eventos já conhecida (JDK 1.1). Utilizar também características do *Java Activation Framework* (JAF), como por exemplo no caso de acesso a dados;
- Classes modulares, facilitando a adicão de funções básicas, bem como a manipulação de correio eletrônico a qualquer aplicação;
- Apoio ao desenvolvimento de aplicações com suporte a correio eletrônico de forma robusta, as quais podem manipular uma variedade de formatos complexos de mensagens de correio, tipos de dados, e protocolos de acesso e transporte.

A API JavaMail tem muitas características dos protocolos padrão como o IMAP, MAPI, CMC, C-Client e outras APIs e sistemas de mensagens; muitos dos conceitos presentes nestes outros sistemas, também estão presentes na API JavaMail.

Esta API é fácil de usar porque ela utiliza as características da linguagem Java, não disponíveis em outras APIs, e porque ela usa o modelo de objetos Java para proteger as aplicações das complexidades de implementação.

A API JavaMail está em conformidade com os padrões correntes, ao mesmo tempo em que não está limitada aos padrões correntes, os quais sufocam as inovações futuras. Encontramos nesta especificação o balanceamento ideal para os nosso objetivos entre a simplicidade e a sofisticação: o simples é fácil de atingir e o sofisticado é plenamente possível.

JavaMail suporta:

- Diferentes implementações de sistemas de mensagens;
- Diferentes armazenamentos de mensagens;
- Diferentes formatos de mensagens;
- Diferentes formas de transporte das mensagens;
- Implementação de sistemas particulares de mensagens, como: IMAP4, POP3 e SMTP;

A implementação das classes e métodos estritamente necessários para prover a funcionalidade requerida para a implementação (não sobrecarregando o modelo).

Apêndice C

Classe Pasta
Pasta de Classificação Programada - *SFolder*

```
/* Classe Pasta
 * Implementa toda a funcionalidade de pastas automáticas do servidor SPOP
 * Fellipe Araújo Aleixo
 * Laboratório de Sistemas Distribuídos
 * DSC / CCT / UFPB
 */

package spop;

import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.mail.*;
import javax.mail.search.*;
import javax.mail.internet.*;

public class Pasta {
  String Name;     // Nome de referência da pasta
  Vector Rules;    // Vetor que representa a regra de formação da pasta
  Vector Messages; // Vetor com a lista das mensagens que compõem a pasta

  public Pasta create (Store store, String name, String rule) {
    Pasta Resultado = new Pasta();

    // Nome da pasta que está sendo criada
    Resultado.Name = name;

    // Análise lexo-sintática para verificação da validade da regra de formação
    // Separação da regra de formação em unidades de critério
    if (rule.equals("")) {
      return(Resultado);
    } else {
      StringTokenizer tokens = new StringTokenizer(rule);
      String token;
      // P A R S E R
      while (tokens.hasMoreElements()) {
        token = (String) tokens.nextElement();
        // Fase (A) do Parser - Captura do critério
        if (token.equals("bcc")) {                  // BCC
          Resultado.Rules.addElement(novotexto(tokens, "bcc"));
        } else if (token.equals("body")) {          // BODY
          Resultado.Rules.addElement(novotexto(tokens, "body"));
        } else if (token.equals("cc")) {            // CC
          Resultado.Rules.addElement(novotexto(tokens, "cc"));
        } else if (token.equals("from")) {          // FROM
          Resultado.Rules.addElement(novotexto(tokens, "from"));
        } else if (token.equals("subject")) {       // SUBJECT
          Resultado.Rules.addElement(novotexto(tokens, "subject"));
        } else if (token.equals("text")) {          // TEXT
          Resultado.Rules.addElement(novotexto(tokens, "text"));
        } else if (token.equals("to")) {            // TO
          Resultado.Rules.addElement(novotexto(tokens, "to"));
        } else if (token.equals("message")) {       // MESSAGE SET
          if (tokens.hasMoreElements()) {
            String nexttoken = (String) tokens.nextElement();
            if (nexttoken.equals("set")) {
              Resultado.Rules.addElement(novonumerico(tokens, "message set"));
            }
          }
        } else if (token.equals("size")) {          // SIZE
```

```
        Resultado.Rules.addElement(novonumerico(tokens, "size"));
      } else if (token.equals("uid")) {          // UID
        Resultado.Rules.addElement(novonumerico(tokens, "uid"));
      } else if (token.equals("sent")) {         // SENT DATE
        if (tokens.hasMoreElements()) {
          String nexttoken = (String) tokens.nextElement();
          if (nexttoken.equals("date")) {
            Resultado.Rules.addElement(novodata(tokens, "sent date"));
          }
        }
      } else if (token.equals("receive")) {      // RECEIVE DATE
        if (tokens.hasMoreElements()) {
          String nexttoken = (String) tokens.nextElement();
          if (nexttoken.equals("date")) {
            Resultado.Rules.addElement(novodata(tokens, "receive date"));
          }
        }
      } else if (token.equals("with")) {         // WITH FLAGS
        if (tokens.hasMoreElements()) {
          String nexttoken = (String) tokens.nextElement();
          if (nexttoken.equals("flags")) {
            Resultado.Rules.addElement(novoflags(tokens));
          }
        }
      } else if (token.startsWith("not")) {       // NOT((A))
        if (token.charAt(3) == '(') {
        // Não implementado no momento...
        }

      // Captura do operador
      } else if (token.equals("||")) {
        Operador novooperador = new Operador();
        novooperador.operador = "||";
        Resultado.Rules.addElement(novooperador);
      } else if (token.equals("&&")) {
        Operador novooperador = new Operador();
        novooperador.operador = "&&";
        Resultado.Rules.addElement(novooperador);
      }
    }
  }
  // Efetuar busca individual por cada critério na caixa de correio do usuário
  Resultado.Messages.addElement(Search(store, Rules));

  // Retorno da pasta criado
  return(Resultado);
}

// Captura de um critério de texto
private Criterio novotexto(StringTokenizer tokens, String campo) {

  Criterio novocriterio = new Criterio();

  if (tokens.hasMoreElements()) {
    String nexttoken = (String) tokens.nextElement();
    if (nexttoken.equals("with")) {
      if (tokens.hasMoreElements()) {
        String cadeia = (String) tokens.nextElement();
        if (cadeia.startsWith("\"")) {
          if (cadeia.endsWith("\"")) {
            int tamanho = cadeia.length()-1;
            String valor = cadeia.substring(1, tamanho);
            // Adiciona no vetor de critérios
            novocriterio.Classe = 't';
            novocriterio.Campo = campo;
            novocriterio.Valor = valor;
          }
        }
      }
    }
  }
  return(novocriterio);
}

// Captura de um critério numérico
private Criterio novonumerico(StringTokenizer tokens, String campo) {
```

```
  Criterio novocriterio = new Criterio();

  if (tokens.hasMoreElements()) {
    String comparacao = "";

    String nexttoken = (String) tokens.nextElement();
    if (nexttoken.equals(">")) {
      comparacao = "maior que";
    } else if (nexttoken.equals("<")) {
      comparacao = "menor que";
    } else if (nexttoken.equals("=")) {
      comparacao = "igual";
    } else if (nexttoken.equals(">=")) {
      comparacao = "maior igual";
    } else if (nexttoken.equals("<=")) {
      comparacao = "menor igual";
    } else if (nexttoken.equals("!=")) {
      comparacao = "diferente";
    } else if (nexttoken.equals("in")) {
      if (tokens.hasMoreElements()) {
        String limiteinferior = (String) tokens.nextElement();
        if (tokens.hasMoreElements()) {
          String nextone = (String) tokens.nextElement();
          if (nextone.equals("..")) {
            if (tokens.hasMoreElements()) {
              String limitesuperior = (String) tokens.nextElement();
              novocriterio.Classe = 'n';
              novocriterio.intervalo = true;
              Integer inferior = new Integer(limiteinferior);
              Integer superior = new Integer(limitesuperior);
              novocriterio.limiteinferior = inferior.intValue();
              novocriterio.limitesuperior = superior.intValue();
            }
          }
        }
      }
    }
    if (tokens.hasMoreElements()) {
      String valor = (String) tokens.nextElement();
      novocriterio.Classe = 'n';
      novocriterio.Comparacao = comparacao;
      Integer valorinteiro = new Integer(valor);
      novocriterio.valornumerico = valorinteiro.intValue();
    }
  }
  return(novocriterio);
}

// Captura de um critério de datas
private Criterio novodata(StringTokenizer tokens, String campo) {

  Criterio novocriterio = new Criterio();

  if (tokens.hasMoreElements()) {
    String comparacao = new String();

    String nexttoken = (String) tokens.nextElement();
    if (nexttoken.equals(">")) {
      comparacao = "maior que";
    } else if (nexttoken.equals("<")) {
      comparacao = "menor que";
    } else if (nexttoken.equals("=")) {
      comparacao = "igual";
    } else if (nexttoken.equals(">=")) {
      comparacao = "maior igual";
    } else if (nexttoken.equals("<=")) {
      comparacao = "menor igual";
    } else if (nexttoken.equals("!=")) {
      comparacao = "diferente";
    }
    if (tokens.hasMoreElements()) {
      Calendar datapesquisa = quebradata((String)tokens.nextElement());
      novocriterio.Classe = 'd';
      novocriterio.Campo = campo;
      novocriterio.Comparacao = comparacao;
      novocriterio.data = datapesquisa;
    }
```

```
    }
    return(novocriterio);
  }

  // Quebra o String simbolizando a data em uma classe Calendar
  private Calendar quebradata(String data) {
    Calendar dataretorno = Calendar.getInstance();
    Integer dia = new Integer(data.substring(0, 2));
    Integer mes = new Integer(data.substring(3, 5));
    Integer ano = new Integer(data.substring(6));

    dataretorno.set(ano.intValue(),mes.intValue(),dia.intValue());
    return(dataretorno);
  }

  // Captura de um critério de flags
  private Criterio novoflags(StringTokenizer tokens) {

    Criterio novocriterio = new Criterio();

    if (tokens.hasMoreElements()) {
      String nexttoken = (String) tokens.nextElement();
      if (nexttoken.equals("(")) {
        Vector flags = new Vector();
        while (tokens.hasMoreElements()) {
          String flag = (String) tokens.nextElement();
          if (flag.equals("\\answered")) {
            flags.addElement(flag);
          } else if (flag.equals("\\deleted")) {
            flags.addElement(flag);
          } else if (flag.equals("\\draft")) {
            flags.addElement(flag);
          } else if (flag.equals("\\flagged")) {
            flags.addElement(flag);
          } else if (flag.equals("\\recent")) {
            flags.addElement(flag);
          } else if (flag.equals("\\seen")) {
            flags.addElement(flag);
          } else if (flag.equals(")")) {
            break;
          }
        }
        // Monta o critério de flags e adiciona no vetor de regras
        novocriterio.Classe = 'f';
        novocriterio.flags = flags;
      }
    }
    return(novocriterio);
  }

  // Busca em uma caixa de correio por mensagens que satisfaçam tais regras
  public Vector Search(Store store, Vector regras) {
    Vector resultado = new Vector();

    // A busca nas mensagens é feita sequencialmente critério a critério
    int indice = 0;
    Stack parciais = new Stack();
    Object elemento = new Object();
    Vector resultadoparcial = new Vector();
    Vector registrador1 = new Vector();
    Vector registrador2 = new Vector();

    while (indice < regras.size()) {
      // Enquanto houver regras repete operação
      elemento = regras.elementAt(indice++);

      if (elemento instanceof Operador) {
        Operador operador = (Operador) elemento;
        // Se encontrar um operador AND empilha o resultado parcial
        // Isto acontece, pois o OU tem precedência ao AND
        if (operador.operador.equals("&&")) {
          parciais.push(resultadoparcial);
          resultadoparcial.removeAllElements();
        }
      } else if (elemento instanceof Criterio) {
        Criterio criterio = (Criterio) elemento;
        if (criterio.Classe == 't') {
```

```
          // Search TEXTO
          registrador1 = SearchTexto(store, criterio);
        } else if (criterio.Classe == 'n') {
          // Search NUMÉRICO
          registrador1 = SearchNumerico(store, criterio);
        } else if (criterio.Classe == 'd') {
          // Search DATA
          registrador1 = SearchData(store, criterio);
        } else if (criterio.Classe == 'f') {
          // Search FLAGS
          registrador1 = SearchFlags(store, criterio);
        }
        // Se o registrador estiver vazio, armazena o resultado parcial nele
        if (resultadoparcial.isEmpty()) {
          resultadoparcial = (Vector) registrador1.clone();
          registrador1.removeAllElements();
        // Se já houver resultado parcial, efetua operação com registrador1
        } else {
          registrador2 = operacaoOU(registrador1, resultadoparcial);
          resultadoparcial = (Vector) registrador2.clone();
          registrador1.removeAllElements();
          registrador2.removeAllElements();
        }
      }
      // Acabaram os elementos que compõem a regra de foramação da pasta
      // Efetuar agora as operações AND
      while (!parciais.empty()) {
        registrador1 = (Vector) parciais.pop();
        registrador2 = operacaoAND(resultadoparcial, registrador1);
        resultadoparcial = (Vector) registrador2.clone();
        registrador1.removeAllElements();
        registrador2.removeAllElements();
      }
      resultado = resultadoparcial;
    }
    return(resultado);
  }

  //
  // SEARCH TEXTO
  // ------ -----
  private Vector SearchTexto(Store store, Criterio criterio) {
    Vector resultado = new Vector();

    if (store.isConnected()) {
      try {
        Folder folder = store.getDefaultFolder();

        if (folder != null) {
          int totalMessages = folder.getMessageCount();

          // Varer mensagem a mensagem da pasta do cliente
          int indice = 0;
          while (indice < totalMessages) {
            Message mensagem = folder.getMessage(indice);

            // Se o critério abranger o BCC
            if (criterio.Campo.equals("bcc")) {
              RecipientStringTerm padrao = new
RecipientStringTerm(Message.RecipientType.BCC,criterio.Valor);
              if (padrao.match(mensagem)) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            }
            // Se o critério abranger o BODY
            else if (criterio.Campo.equals("body")) {
              BodyTerm padrao = new BodyTerm(criterio.Valor);
              if (padrao.match(mensagem)) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            }
            // Se o critério abranger o CC
            else if (criterio.Campo.equals("cc")) {
              RecipientStringTerm padrao = new
RecipientStringTerm(Message.RecipientType.CC,criterio.Valor);
```

```
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }
          // Se o critério abranger o FROM
          else if (criterio.Campo.equals("from")) {
            FromStringTerm padrao = new FromStringTerm(criterio.Valor);
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }
          // Se o critério abranger o SUBJECT
          else if (criterio.Campo.equals("subject")) {
            SubjectTerm padrao = new SubjectTerm(criterio.Valor);
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }
          // Se o critério abranger o TEXT
          else if (criterio.Campo.equals("text")) {
            HeaderTerm padrao1 = new HeaderTerm("",criterio.Valor);
            BodyTerm padrao2 = new BodyTerm(criterio.Valor);
            MessageIDTerm padrao3 = new MessageIDTerm(criterio.Valor);
            if ((padrao1.match(mensagem)) || (padrao2.match(mensagem))) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }
          // Se o critério abranger o TO
          else if (criterio.Campo.equals("to")) {
            RecipientStringTerm padrao = new
RecipientStringTerm(Message.RecipientType.TO,criterio.Valor);
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }
          indice++;
        }
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
  return(resultado);
}

//
// SEARCH NUMÉRICO
// ------ --------
private Vector SearchNumerico(Store store, Criterio criterio) {
  Vector resultado = new Vector();

  if (store.isConnected()) {
    try {
      Folder folder = store.getDefaultFolder();
      if (folder != null) {
        int totalMessages = folder.getMessageCount();

        // Estipulando o tipo de comparação
        int comparacao = 0;
        if (criterio.Comparacao.equals(">")) {
          comparacao = ComparisonTerm.GT;
        } else if (criterio.Comparacao.equals("<")) {
          comparacao = ComparisonTerm.LT;
        } else if (criterio.Comparacao.equals(">=")) {
          comparacao = ComparisonTerm.GE;
        } else if (criterio.Comparacao.equals("<=")) {
          comparacao = ComparisonTerm.LE;
        } else if (criterio.Comparacao.equals("=")) {
          comparacao = ComparisonTerm.EQ;
        } else if (criterio.Comparacao.equals("!=")) {
          comparacao = ComparisonTerm.NE;
        }
```

```
        // Varer mensagem a mensagem da pasta do cliente
        int indice = 0;
        while (indice < totalMessages) {
          Message mensagem = folder.getMessage(indice);

          // Quando o critério se referir ao MESSAGE SET
          if (criterio.Campo.equals("message set")) {
            if (criterio.Comparacao.equals(">")) {
              if (mensagem.getMessageNumber() > criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            } else if (criterio.Comparacao.equals("<")) {
              if (mensagem.getMessageNumber() < criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            } else if (criterio.Comparacao.equals(">=")) {
              if (mensagem.getMessageNumber() >= criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            } else if (criterio.Comparacao.equals("<=")) {
              if (mensagem.getMessageNumber() <= criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            } else if (criterio.Comparacao.equals("=")) {
              if (mensagem.getMessageNumber() == criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            } else if (criterio.Comparacao.equals("!=")) {
              if (mensagem.getMessageNumber() != criterio.valornumerico) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            }
          }

          // Quando o critério de referir ao SIZE
          else if (criterio.Campo.equals("size")) {
            SizeTerm padrao = new SizeTerm(comparacao,criterio.valornumerico);
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }
          }

        }
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
  return(resultado);
}

//
// SEARCH DATA
// ------ ----
private Vector SearchData(Store store, Criterio criterio) {
  Vector resultado = new Vector();

  if (store.isConnected()) {
    try {
      Folder folder = store.getDefaultFolder();
      if (folder != null) {
        int totalMessages = folder.getMessageCount();

        // Estipulando o tipo de comparação
        int comparacao = 0;
        if (criterio.Comparacao.equals(">")) {
          comparacao = ComparisonTerm.GT;
        } else if (criterio.Comparacao.equals("<")) {
          comparacao = ComparisonTerm.LT;
```

```
          } else if (criterio.Comparacao.equals(">=")) {
            comparacao = ComparisonTerm.GE;
          } else if (criterio.Comparacao.equals("<=")) {
            comparacao = ComparisonTerm.LE;
          } else if (criterio.Comparacao.equals("=")) {
            comparacao = ComparisonTerm.EQ;
          } else if (criterio.Comparacao.equals("!=")) {
            comparacao = ComparisonTerm.NE;
          }

          // Transformando de Caledar para Date para utilizar o método match
          Date dataparapesquisa = new Date(criterio.data.YEAR,
                                           criterio.data.MONTH,
                                           criterio.data.DATE);

          // Varer mensagem a mensagem da pasta do cliente
          int indice = 0;
          while (indice < totalMessages) {
            Message mensagem = folder.getMessage(indice);

            // Quando o critério se referir a SENT DATE
            if (criterio.Campo.equals("sent date")) {
              SentDateTerm padrao = new SentDateTerm(comparacao,dataparapesquisa);
              if (padrao.match(mensagem)) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            }
            // Quando o critério se referir a RECEIVE DATE
            if (criterio.Campo.equals("receive date")) {
              ReceivedDateTerm padrao = new
ReceivedDateTerm(comparacao,dataparapesquisa);
              if (padrao.match(mensagem)) {
                String[] id = mensagem.getHeader("Message-ID");
                resultado.addElement(id[0]);
              }
            }

          }
        }
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
    return(resultado);
  }

  //
  // SEARCH FLAGS
  // ------ -----
  private Vector SearchFlags(Store store, Criterio criterio) {
    Vector resultado = new Vector();

    if (store.isConnected()) {
      try {
        Folder folder = store.getDefaultFolder();
        if (folder != null) {
          int totalMessages = folder.getMessageCount();

          // Criando a estrutura flags para servirem de parâmetros para a busca
          Flags flags = new Flags();
          int numerodeflags = criterio.flags.size();
          int posicao = 0;
          while (posicao < numerodeflags) {
            if (criterio.flags.elementAt(posicao).equals("\\ANSWERED")) {
              flags.add(Flags.Flag.ANSWERED);
            } else if (criterio.flags.elementAt(posicao).equals("\\DELETED")) {
              flags.add(Flags.Flag.DELETED);
            } else if (criterio.flags.elementAt(posicao).equals("\\DRAFT")) {
             flags.add(Flags.Flag.DRAFT);
            } else if (criterio.flags.elementAt(posicao).equals("\\FLAGGED")) {
              flags.add(Flags.Flag.FLAGGED);
            } else if (criterio.flags.elementAt(posicao).equals("\\RECENT")) {
              flags.add(Flags.Flag.RECENT);
            } else if (criterio.flags.elementAt(posicao).equals("\\SEEN")) {
              flags.add(Flags.Flag.SEEN);
            }
```

```
            posicao++;
          }

          // Varer mensagem a mensagem da pasta do cliente
          int indice = 0;
          while (indice < totalMessages) {
            Message mensagem = folder.getMessage(indice);

            // Caso os referidos FLAGS encontrados na estrutura FLAGS estiverem
            // setados (como TRUE) nas mensagens, tais mensagens fazem parte do
            // resultado.
            FlagTerm padrao = new FlagTerm(flags,true);
            if (padrao.match(mensagem)) {
              String[] id = mensagem.getHeader("Message-ID");
              resultado.addElement(id[0]);
            }

          }
        }
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
    return(resultado);
  }

  // *** Operações OU e AND
  // Operação OU
  private Vector operacaoOU(Vector conjunto1, Vector conjunto2) {
    Vector resultado = new Vector();

    resultado = (Vector) conjunto1.clone();
    int indice = 0;
    while (indice < conjunto2.size()) {
      if (resultado.contains(conjunto2.elementAt(indice))) {
        indice = indice + 1;
      } else {
        resultado.addElement(conjunto2.elementAt(indice));
        indice = indice + 1;
      }
    }
    return(resultado);
  }

  // Operação AND
  private Vector operacaoAND(Vector conjunto1, Vector conjunto2) {
    Vector resultado = new Vector();

    int indice = 0;
    while (indice < conjunto1.size()) {
      if (conjunto2.contains(conjunto1.elementAt(indice))) {
        resultado.addElement(conjunto1.elementAt(indice));
        indice = indice + 1;
      } else {
        indice =  indice + 1;
      }
    }
    return(resultado);
  }

}
```

Referências Bibliográficas

[Aboba99] B. Aboba, G. Zorn, "RADIUS Authentication Client MIB.", Junho de 1999 – RFC 2618.

[Ander84] B. Anderson, "TACACS User Identification Telnet Option", 01/12/1984 – RFC 0927.

[Armst98] E. Armstrong, **'JBuilder 2 Bible'**: IDG Books Worldwide, Inc., Foster City, California, 1998.

[Banya98] *"Banyan System Specification for Version 3.5 – Banyan SiteMinder"*, Banyan Systems Incorporated, 1998.

[Boren96] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", 02/12/1996 – RFC RFC 2046.

[Borma93] D. Borman, "Telnet Authentication Option", Fevereiro de 1993 – RFC 1416.

[Bosen96] B. Bosen, *"SafeWord Architecture: A Historical Tour"*, Secure Computing Corporation, Authentication Division, Concord, California, USA, 1996.

[Bussm95] J. E. C. Bussmann, "BART - Uma Biblioteca Orientada a Objetos de Apoio à Recuperação Textual", COPIN / CCT / UFPB, Dezembro de 1995.

[Butle85] M. Butler, D. Chase, J. Goldberger, J. Postel, J. Reynolds, "Post Office Protocol – Version 2", 01/02/1985 – RFC 0937.

[Cargi95] N. Freed, A. Cargille, "SMTP Service Extension for Comand Pipelining", 04/10/1995 – RFC 1854.

[Catap94] **'Microsoft Mail Step by Step'**: Catapult, Inc., Microsoft Press, Novembro de 1994.

[Clark86] D. Clark, M. Lambert, "PCMAIL: A Distributed Mail System for Personal Computers", 01/05/1986 – RFC 0984.

[Clark86-2] D. Clark, M. Lambert, "PCMAIL: A Distributed Mail System for Personal Computers", 01/12/1986 – RFC 0993.

[Compu97] *"Interceptor Version 3.0 – Software Specification"*, Computerlink Control Systems Ltd, Northumberland, England, Julho de 1997.

[Corne97] G. Cornell, C. Horstmann, **'Core Java 1.1: Volume II – Advanced Features'**: Prentice Hall, Inc., Dezembro de 1997.

[Crisp88] M. Crispin, "Interactive Mail Access Protocol: Version 2", 01/07/1988 – RFC 1064.

[Crisp94] M. Crispin, "Distributed Eletronic Models in IMAP4", 20/12/1994 – RFC 1733.

[Crisp94-2] M. Crispin, "IMAP4 Compatibility With IMAP2 and IMAP2BIS", 20/12/1994 – RFC 1732.

[Crisp94-3] M. Crispin, "Internet Message Access Protocol – Version 4", 20/12/1994 – RFC 1730.

[Crisp96] M. Crispin, "Internet Message Access Protocol – Obsolete Syntax", 04/12/1996 – RFC 2062.

[Crisp96-2] M. Crispin, "IMAP4 Compatibility With IMAP2BIS", 05/12/1996 – RFC 2061.

[Crisp96-3] M. Crispin, "Internet Message Access Protocol – Version 4rev1", 04/12/1996 – RFC 2060.

[Crock82] D. Crocker, "Standard for the Format of ARPA Internet Text Messages", 13/08/1982 – RFC 822.

[Crock95] D. Crocker, N. Freed, A. Cargille, "SMTP Service Extension for Checkpoint/Restart", 02/10/1995 – RFC 1845.

[Cyber98] *"CyberSafe – TrustBroker Security Server"*, CyberSafe Corporation, 1998.

[Flynn93] J. Flynn, **'10 Minute Guide to Microsoft Mail for Windows'**: Alpha Books, Inc., Janeiro de 1993.

[Freed94] J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker, "SMTP Service Extension for 8-bit MIMEtransport", 18/07/1994 – RFC 1652.

[Freed95] J. Klensin, N. Freed, K. Moore, "SMTP Service Extension", 06/11/1995 – RFC 1869.

[Freed96] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", 02/12/1996 – RFC 2049.

[Freed96-2] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Messages Bodies", 02/12/1996 – RFC 2045.

[Horst98] C. Horstmann, G. Cornell, **'Core Java 2: Volume I – Fundamentals'**: Prentice Hall, Inc., Dezembro de 1998.

[Hughe97] M. Hughes, C. Hughes, M. Shoffner, M. Winslow, **'JAVA Networking Programming'**: Manning Publications Co., 1997.

[Hughe98] L. Hughes **'Internet e-mail: protocols, standards, and implementation'**: Artech House, Boston, 1988.

[ISO90] ISO/IEC 10021-1:1990 Information technology – Text Communication – Message-Oriented Text Interchange Systems (MOTIS) – Part 1: System and Service Overview.

[ISO90-2] ISO/IEC 10021-3:1990 Information technology – Text Communication – Message-Oriented Text Interchange Systems (MOTIS) – Part 3: Abstract Service Definition Conventions.

[ISO90-3] ISO/IEC 10021-4:1990 Information technology – Text Communication – Message-Oriented Text Interchange Systems (MOTIS) – Part 4: Message Transfer System: Abstract Service Definition and Procedures.

[Jense97] C. Jensen, B. Stone, L. Anderson, **'JBuilder Essentials'**: Osvorne, Inc., Dezembro de 1997.

[Johns85] M. Johns, "Authentication Server", 01/01/1985 – RFC 0931.

[Kanto86] B. Kantor, P. Lapsley, "Network News Transfer Protocol", 01/02/1986 – RFC 0977.

[Kilcu96] B, Kilcullen, **'Introducing Microsoft Exchange'**: Microsoft Press, Abril de 1996.

[Klens94] J. Klensin, N. Freed, K. Moore, "SMTP Service Extension for Message Size Declaration", 18/07/1994 – RFC 1653.

[Klens95] J. Klensin, N. Freed, K. Moore, "SMTP Service Extension for Message Size Declaration", 06/11/1995 – RFC 1870.

[Klens96] N. Freed, J. Klensin, J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", 02/12/1996 – RFC 2048.

[Kohl93] J. Kohl, C. Neuman, "The Kerberos Network Authentication Service (V5)", Setembro de 1993 - RFC 1510.

[Kyber96] *"KyberPASS™ Security Server - KyberPASS™ Dial Access Authentication Server"*, KyberPASS Corporation, Fevereiro de 1996.

[Lambe88] M. Lambert, "PCMAIL: A Distributed Mail System for Personal Computers", 01/06/1988 – RFC 1056.

[Mead92] C. T. Meadow, **'Text Information Retrieval Systems'**: Academic Press, San Diego, California, 1992.

[Moore96] K. Moore, "Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text", 02/12/1996 – RFC 2047.

[Moore96-2] K. Moore, "SMTP Service Extension for Delivery Status Notifications", 15/01/1996 – RFC 1891.

[Myers94] J. Myers, M. Rose, "Post Office Protocol – Vresion 3", 23/11/1994 – RFC 1725.

[Myers94-2] J. Myers, "IMAP4 Authentication Mechanisms", 20/12/1994 – RFC 1731.

[Myers96] J. Myers, M. Rose, "Post Office Protocol – Version 3", 14/05/1996 – RFC 1939.

[Myers97] J. Myers, "IMAP4 QUOTA Extension", Janeiro de 1997 – RFC 2087.

[Myers97-2] J. Myers, "IMAP4 ACL Extension", Janeiro de 1997 – RFC 2086.

[Myers97-3] J. Myers, "IMAP4 Non-synchronizing Literals", Janeiro de 1997 – RFC 2088.

[Nelso96] S. Nelson, **'Field Guide to Microsoft Exchange'**: Microsoft Press, Agosto de 1996.

[Nelso97] S. Nelson, C. Parks, Mitra, "The Model Primary Content Type for Multipurpose Internet Mail Extensions", 10/01/1997 – RFC 2077.

[Notes93] *"Lotus Notes Express"*, Lotus Development Coorporation, Cambridge, MA, 1993.

[Open98] *"Security Management Solutions: OpenVision"*, OpenVision Corporatoin, 1998 (Softwares: *OpenV*Gatekeeper 1.0, OpenV*Secure 1.1, OpenV*SecureMax 3.2.2).*

[Pew99] J. Pew, S. Pew, **'Instant Java (Sun Microsystems Press Java Series)'**: 3rd Edition, Prentice Hall Computer Books, Inc., Janeiro de 1999.

[Poste82] J. Postel, "Simple Mail Transfer Protocol", 01/08/1982 – RFC 821.

[Reyno84] J. Reynolds, "Post Office Protocol", 01/10/1984 – RFC 0918.

[Rose88] M. Rose, "Post Office Protocol – Version 3", 01/11/1988 – RFC 1081.

[Rose88-2] M. Rose, "Post Office Protocol: Version 3: Extended Service Offerings", 01/11/1988 – RFC 1082.

[Rose91] M. Rose, "Post Office Protocol – Versoin 3", 14/05/1991 – RFC 1225.

[Rose93] M. Rose, "Post Office Protocol – Version 3", 16/06/1993 – RFC 1460.

[Rose94] J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker, "SMTP Service Extensions", 18/07/1994 – RFC 1651.

[Showa98] T. Showalter, *"Sieve – A Mail Filtering Language"*, Carnigie Mellon, Agosto de 1998 (Internet Draft: Sieve).

[Sridh97] P. Sridharan, **'Advanced JAVA Networking'**: Prentice-Hall, Inc., New Jersey, 1997.

[Steve94] W. Stevens **'TCP/IP Illustrated, Volume 1 – The Protocols'**: Addisson-Wesley, Reading, Massachusetts, 1994.

[Sun97] *"JavaMail API Design Specification – Version 1.0"*, Sun Mirosystems, Inc., Mountain View, California, 09 de Dezembro de 1997.

[Sun98] *"JavaMail Guide for Service Providers"*, Sun Microsystems, Inc., Palo Alto, California, Agosto de 1998.

[Tanen88] A. S. Tanenbaum **'Computer Networks'**: Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988.

[Vaudr95] G. Vaudreuil, "SMTP Service Extensions for Transmissions of Large and Binary MIME Messages", 16/08/1995 – RFC 1830.

[Winte96] J. De Winter, "SMTP Service Extension for Remote Message Queue Starting", 14/08/1996 – RFC 1985.

[Wrigh95] G. Wright & W. Stevens **'TCP/IP Illustrated, Volume 2 – The Implementation'**: Addisson-Wesley, Reading, Massachusetts, 1995.

[Zorn99] G. Zorn, B. Aboba, "RADIUS Authentication Server MIB.", Junho de 1999 – RFC 2619.