US 20040199529A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2004/0199529 A1

Clark et al. (43) Pub. Date: Oct. 7, 2004

(54) SYSTEM FOR MANAGING AND ORGANIZING STORED ELECTRONIC MESSAGES

(75) Inventors: **David Morley Clark**, Nelson (CA); **Thomas Wilson Gibson**, Nelson (CA); **David Mitchell Bracewell**, Nelson (CA); **Jeff Toy Tsao Kwan**, Toronto (CA)

Correspondence Address:
OYEN, WIGGS, GREEN & MUTALA
480 - THE STATION
601 WEST CORDOVA STREET
VANCOUVER, BC V6B 1G1 (CA)

(73) Assignee: **Caelo Software Inc.**, Nelson (CA)

(21) Appl. No.: **10/817,183**

(22) Filed: **Apr. 5, 2004**

Related U.S. Application Data

(63) Continuation of application No. 09/704,199, filed on Oct. 31, 2000, now Pat. No. 6,725,228.

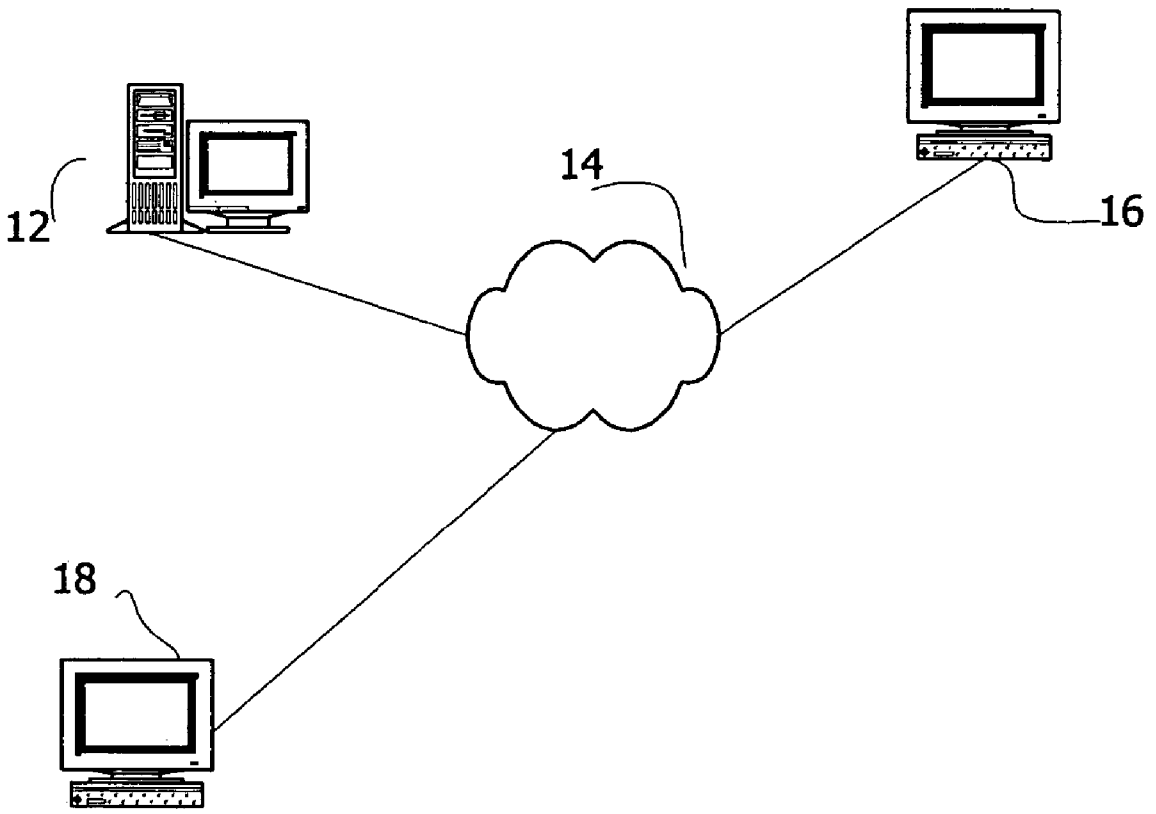
Publication Classification

(51) **Int. Cl.**[7] **G06F 7/00**

(52) **U.S. Cl.** **707/100**

(57) **ABSTRACT**

A computer-based system catalogs and retrieves electronic messages saved in a message store. The system automatically organizes each saved message into multiple folders based on the contents and attributes of the message, and implements improved methods for manually organizing messages. Unlike prior art systems, where a message exists in only one folder, the system uses lightweight message shortcuts to display the message in multiple folders simultaneously. The system preferably permits messages to be organized by: 1) basic message and attachment properties, e.g. date, status, attachment type; 2) extended message properties that the user can specify, e.g. keywords; and 3) correspondent or bulk mail sender/recipient, with automatic separation of bulk mail from correspondence. Performance and usability are improved by performing reads incrementally and by providing multiple sorting keys.



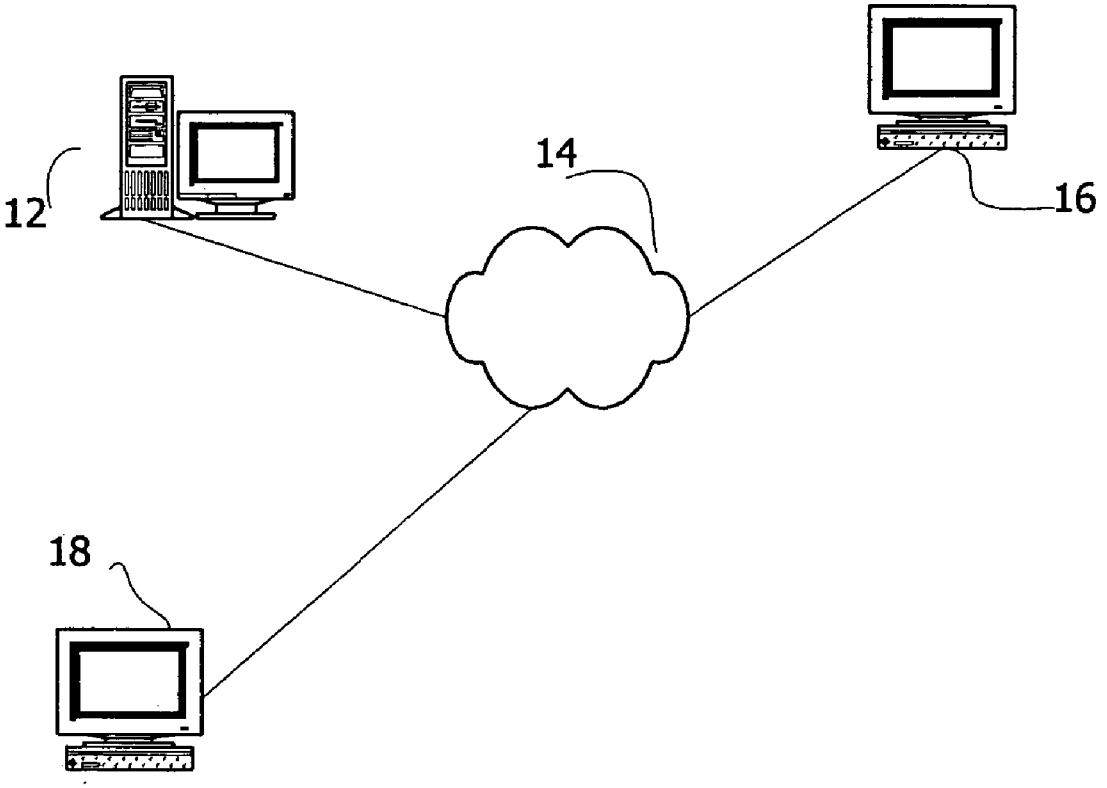


FIG. 1A

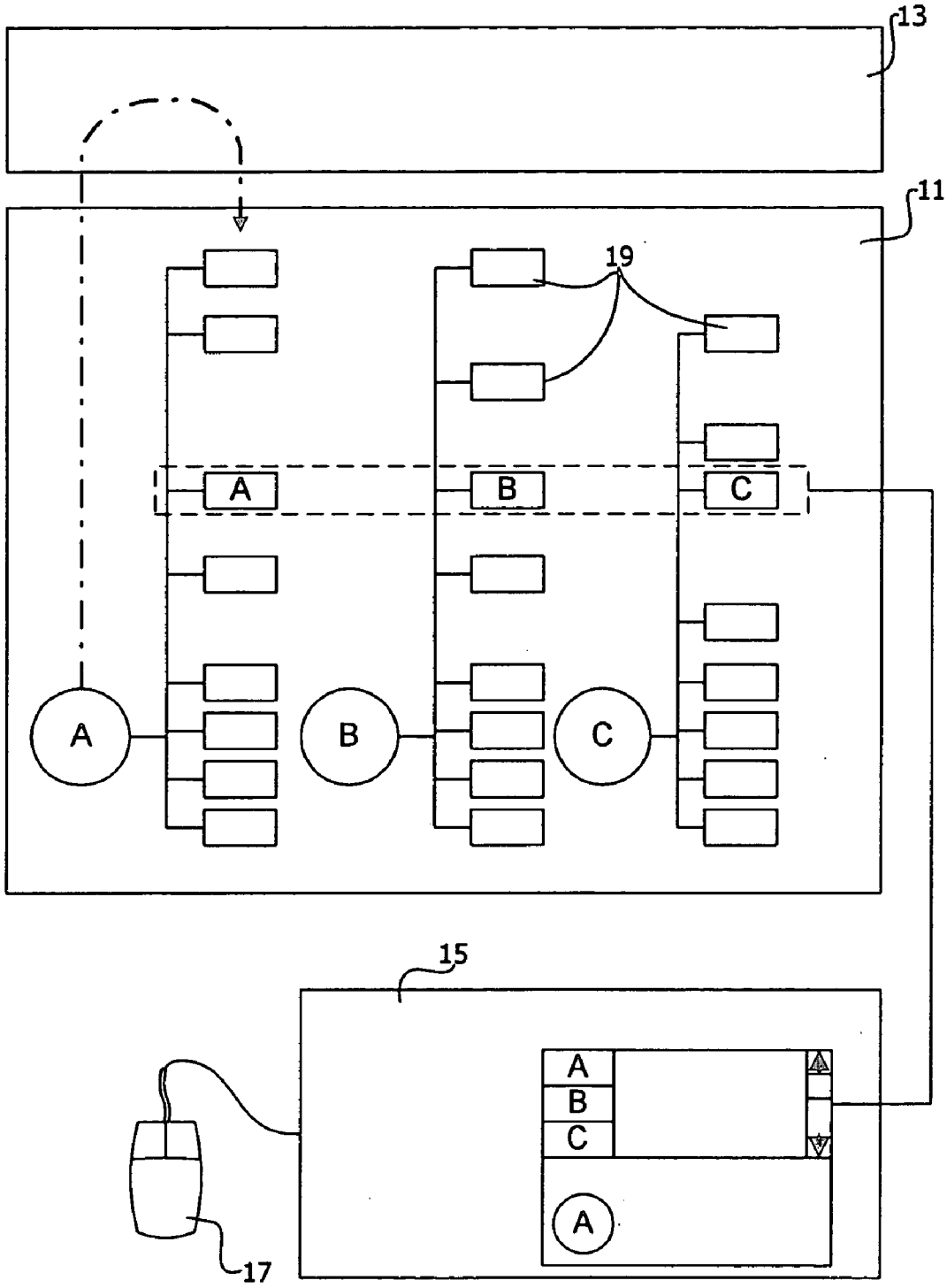


FIG. 1B

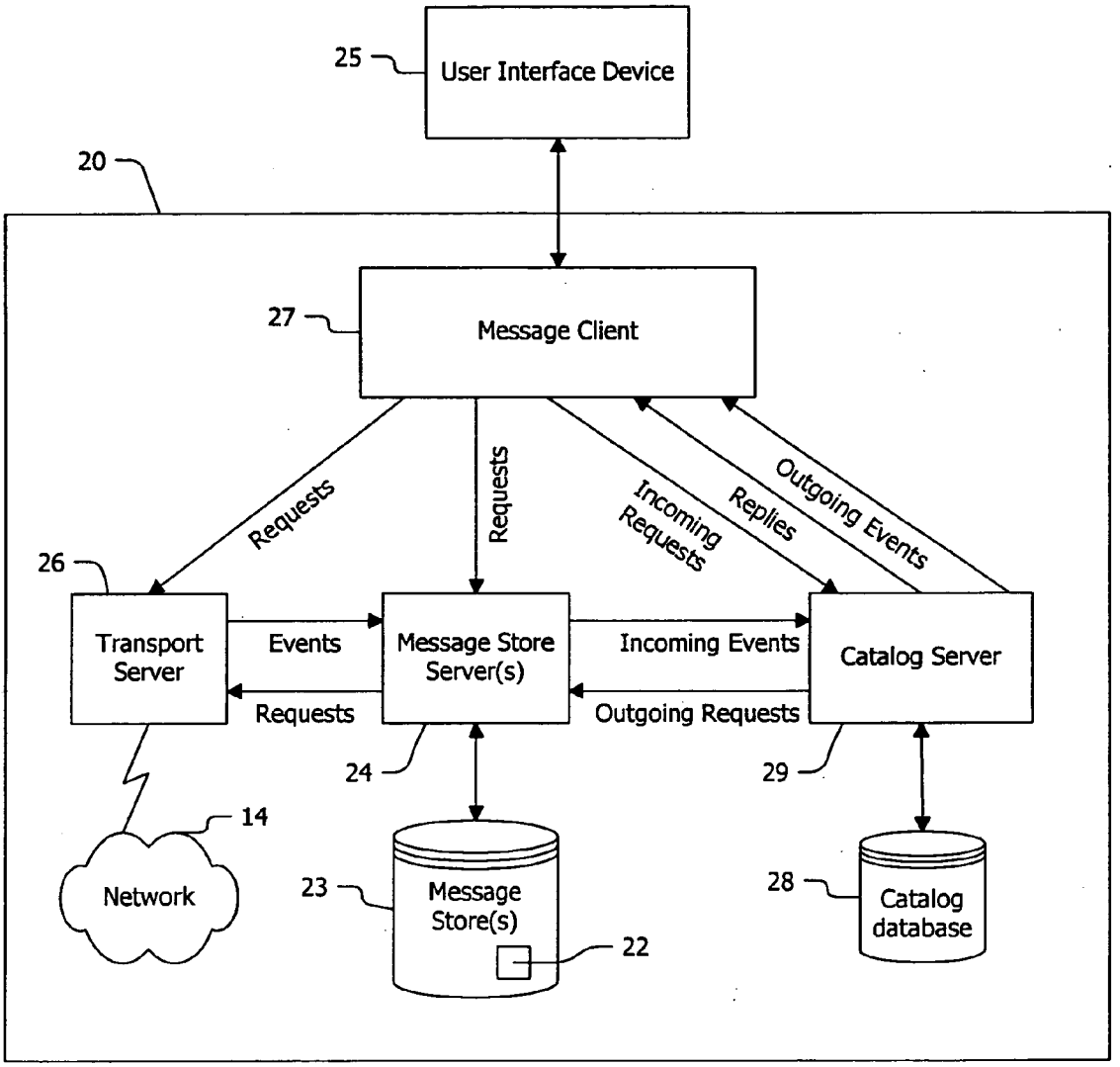

25
User Interface Device
20
27
Message Client
Requests
Requests
Incoming Requests
Replies
Outgoing Events
26
Transport Server
Events
Requests
Message Store Server(s)
Incoming Events
Outgoing Requests
Catalog Server
24
29
14
Network
23
Message Store(s)
22
28
Catalog database

FIG. 2

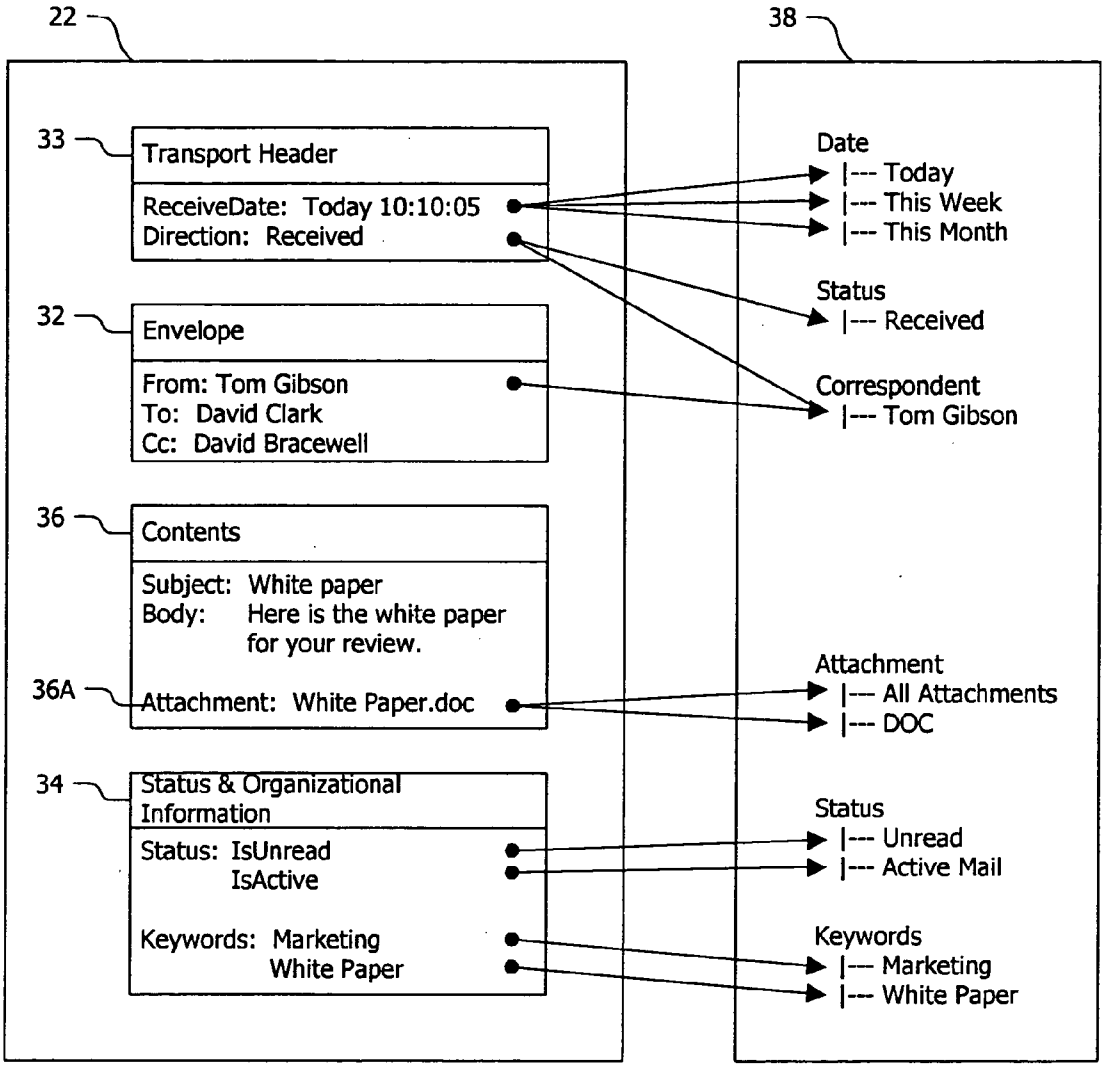

22
38
33
Transport Header
ReceiveDate: Today 10:10:05
Direction: Received
32
Envelope
From: Tom Gibson
To: David Clark
Cc: David Bracewell
36
Contents
Subject: White paper
Body: Here is the white paper for your review.
36A
Attachment: White Paper.doc
34
Status & Organizational Information
Status: IsUnread
IsActive
Keywords: Marketing
White Paper
Date
|--- Today
|--- This Week
|--- This Month
Status
|--- Received
Correspondent
|--- Tom Gibson
Attachment
|--- All Attachments
|--- DOC
Status
|--- Unread
|--- Active Mail
Keywords
|--- Marketing
|--- White Paper

FIG. 3

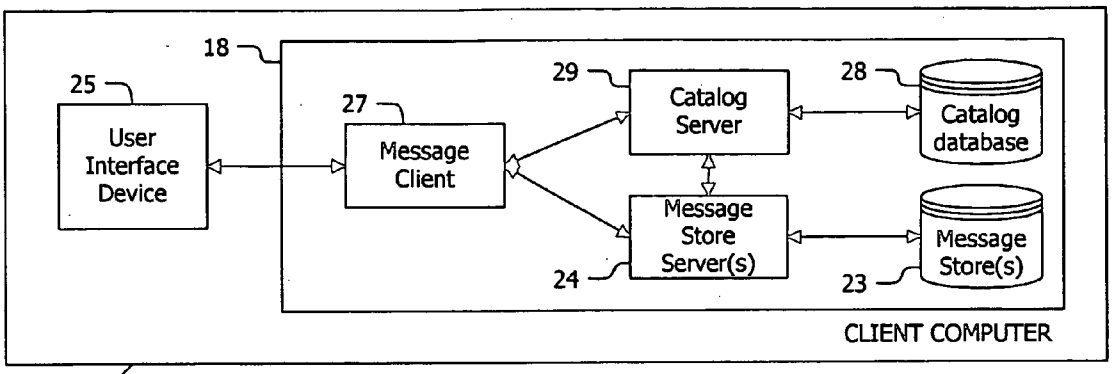


FIG. 4A

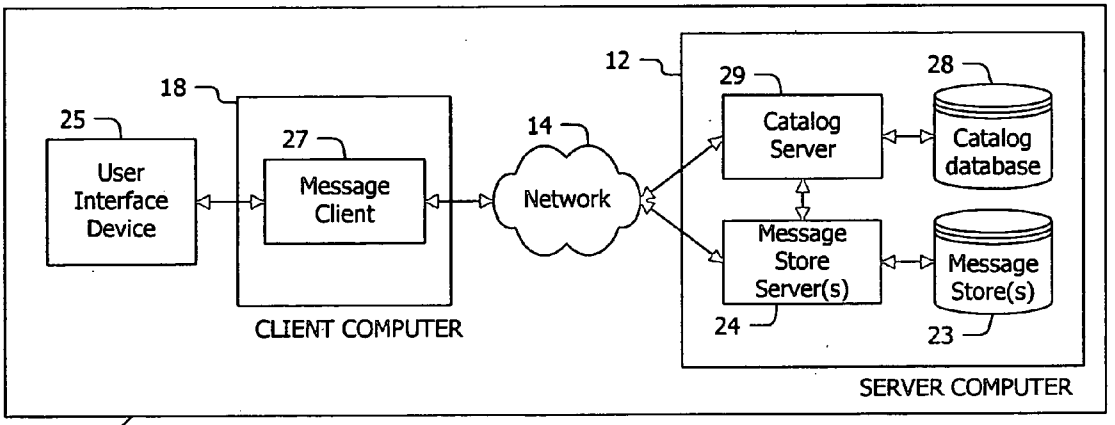


FIG. 4B

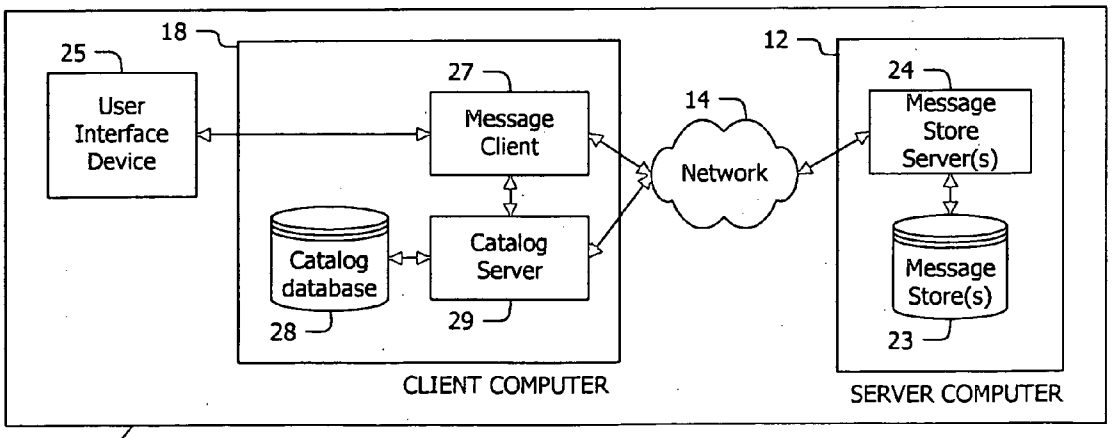


FIG. 4C

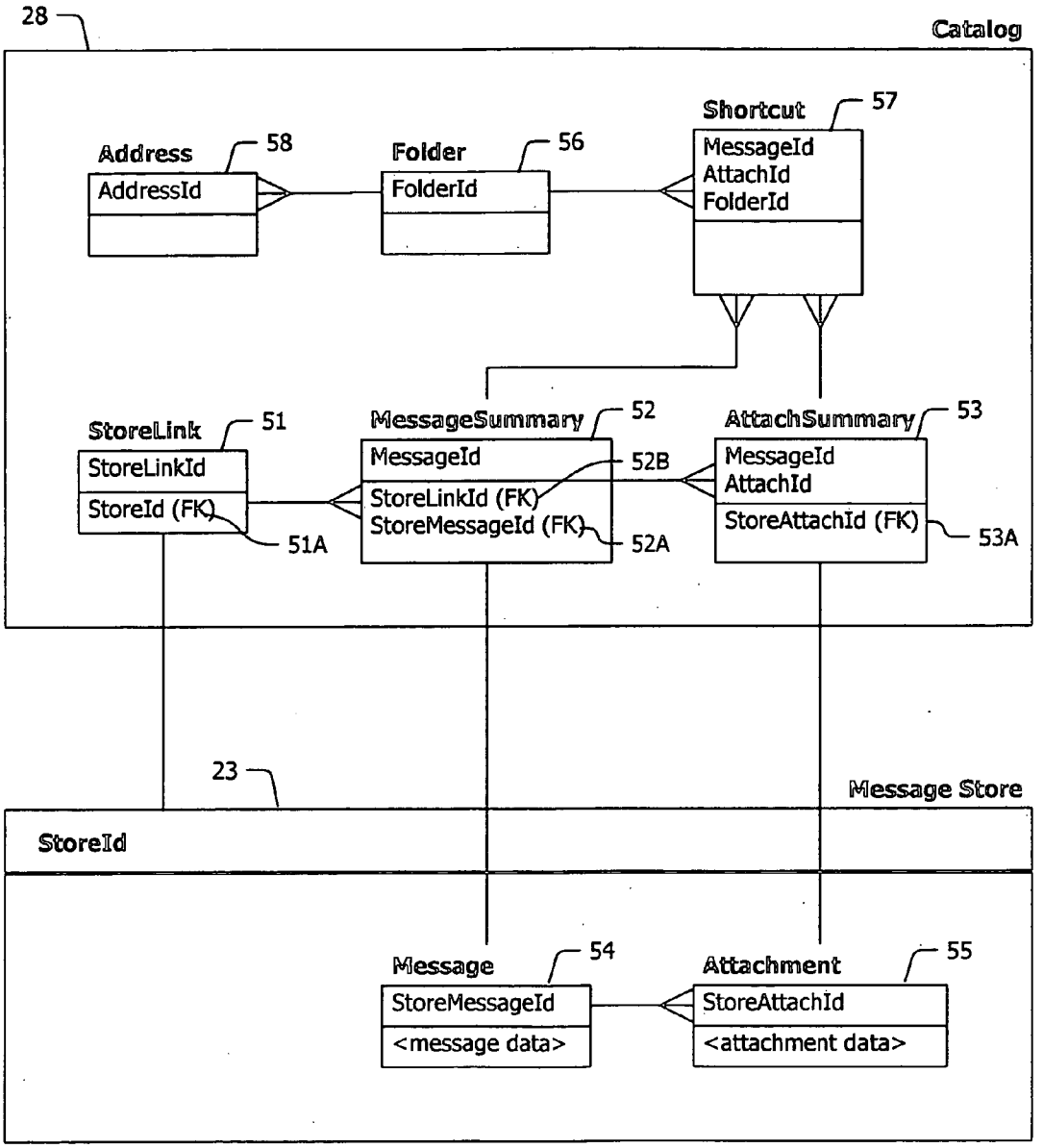

28
Catalog
Shortcut
57
Address
58
Folder
56
MessageId
AttachId
FolderId
AddressId
FolderId
StoreLink
51
MessageSummary
52
AttachSummary
53
StoreLinkId
StoreId (FK)
MessageId
StoreLinkId (FK)
StoreMessageId (FK)
52B
MessageId
AttachId
StoreAttachId (FK)
51A
52A
53A
23
Message Store
StoreId
Message
54
Attachment
55
StoreMessageId
<message data>
StoreAttachId
<attachment data>

FIG. 5A

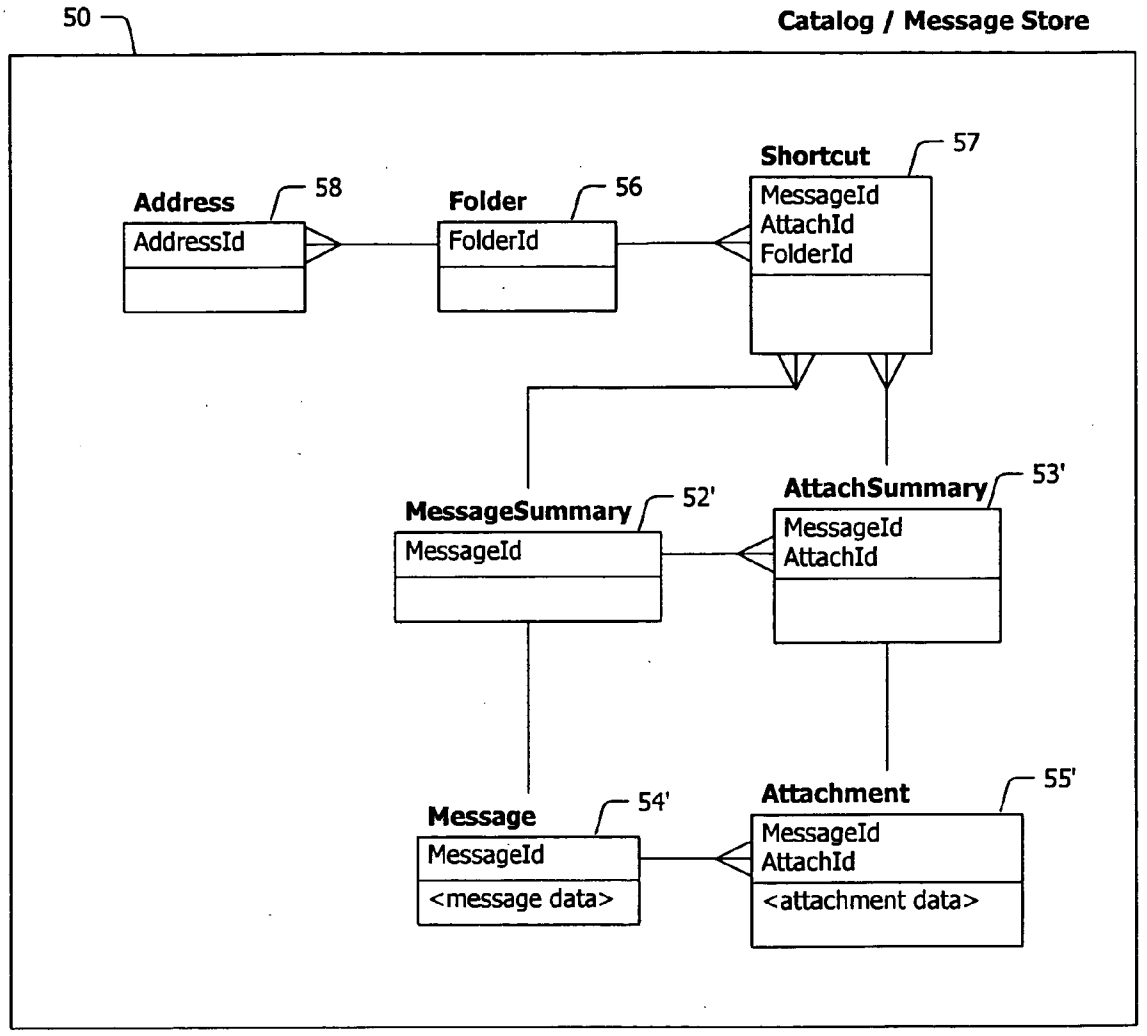

50
Catalog / Message Store
Address
58
AddressId
Folder
56
FolderId
Shortcut
57
MessageId
AttachId
FolderId
MessageSummary
52'
MessageId
AttachSummary
53'
MessageId
AttachId
Message
54'
MessageId
<message data>
Attachment
55'
MessageId
AttachId
<attachment data>

FIG. 5B

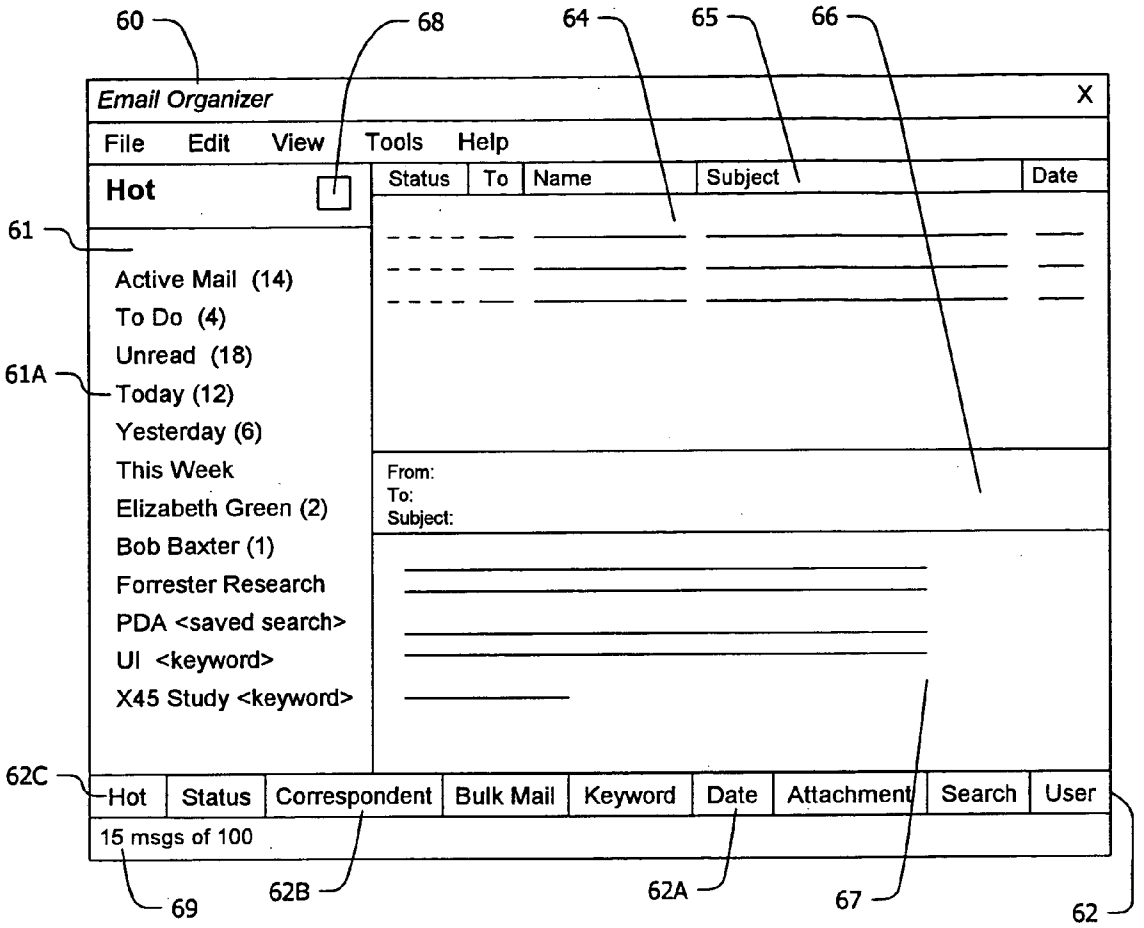

60
68
64
65
66
Email Organizer
X
File
Edit
View
Tools
Help
Hot
Status
To
Name
Subject
Date
61
Active Mail (14)
To Do (4)
Unread (18)
61A
Today (12)
Yesterday (6)
This Week
Elizabeth Green (2)
Bob Baxter (1)
Forrester Research
PDA <saved search>
UI <keyword>
X45 Study <keyword>
From:
To:
Subject:
62C
Hot
Status
Correspondent
Bulk Mail
Keyword
Date
Attachment
Search
User
15 msgs of 100
69
62B
62A
67
62

FIG. 6

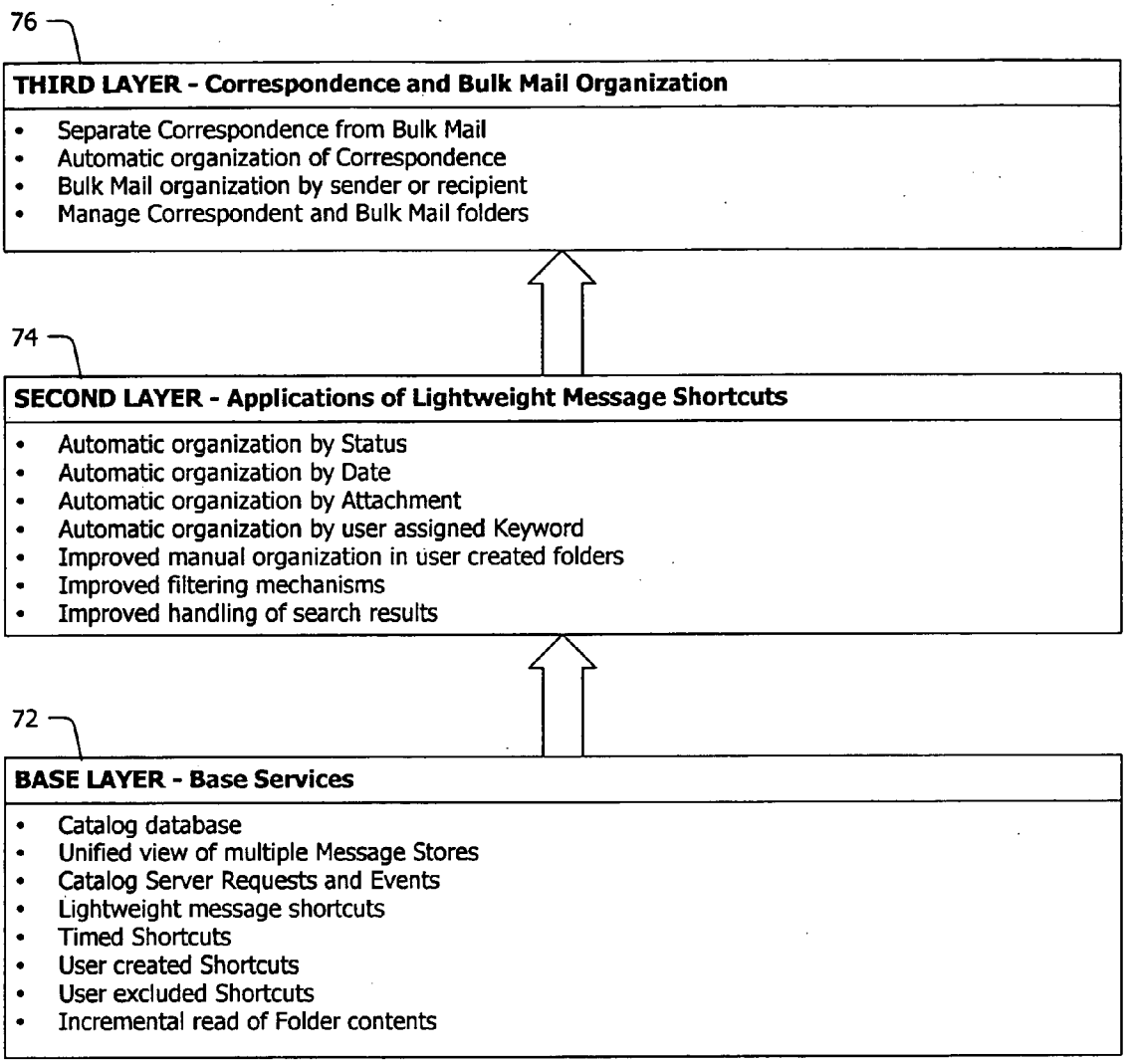

76
THIRD LAYER - Correspondence and Bulk Mail Organization
• Separate Correspondence from Bulk Mail
• Automatic organization of Correspondence
• Bulk Mail organization by sender or recipient
• Manage Correspondent and Bulk Mail folders
74
SECOND LAYER - Applications of Lightweight Message Shortcuts
• Automatic organization by Status
• Automatic organization by Date
• Automatic organization by Attachment
• Automatic organization by user assigned Keyword
• Improved manual organization in user created folders
• Improved filtering mechanisms
• Improved handling of search results
72
BASE LAYER - Base Services
• Catalog database
• Unified view of multiple Message Stores
• Catalog Server Requests and Events
• Lightweight message shortcuts
• Timed Shortcuts
• User created Shortcuts
• User excluded Shortcuts
• Incremental read of Folder contents

FIG. 7

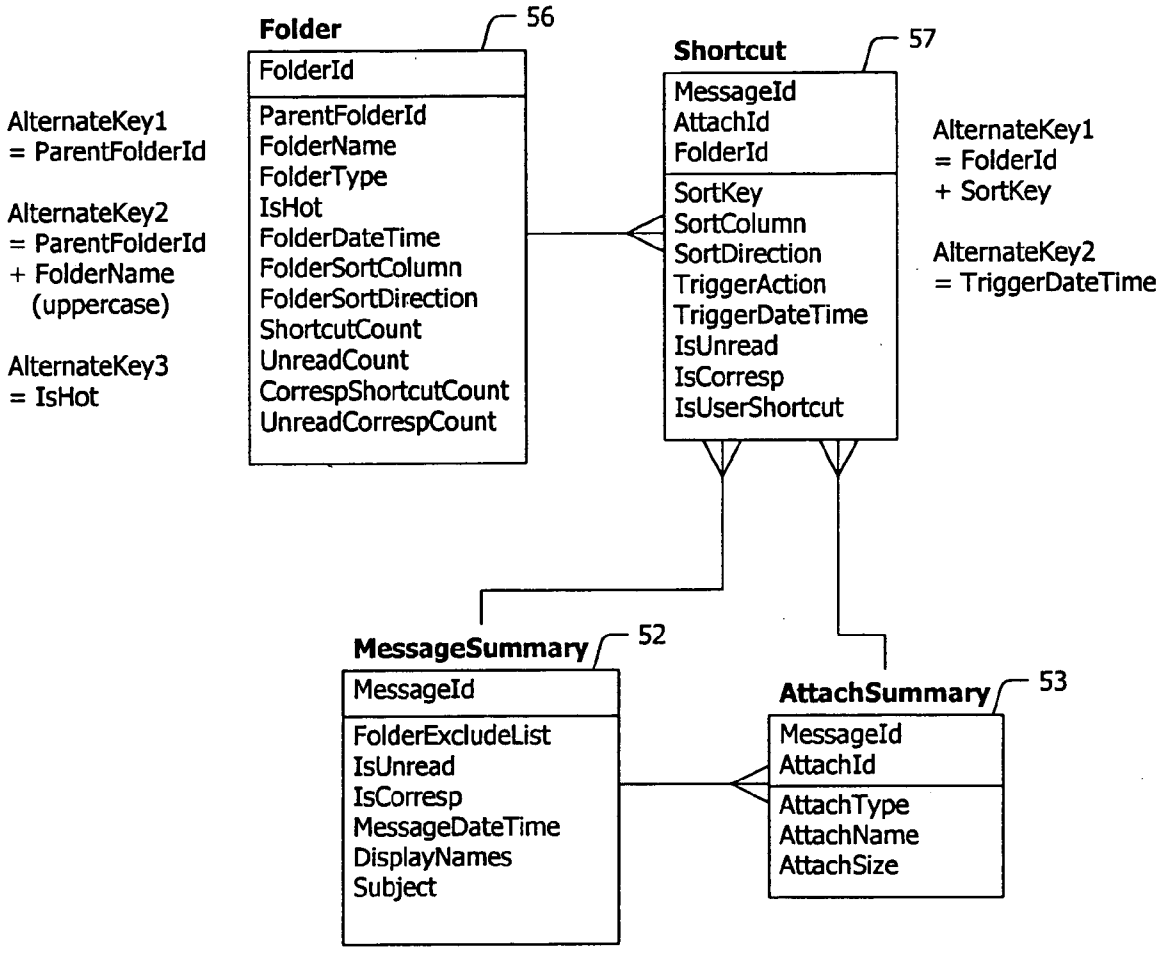

Folder
56
FolderId
ParentFolderId
FolderName
FolderType
IsHot
FolderDateTime
FolderSortColumn
FolderSortDirection
ShortcutCount
UnreadCount
CorrespShortcutCount
UnreadCorrespCount
AlternateKey1
= ParentFolderId
AlternateKey2
= ParentFolderId
+ FolderName
(uppercase)
AlternateKey3
= IsHot
Shortcut
57
MessageId
AttachId
FolderId
SortKey
SortColumn
SortDirection
TriggerAction
TriggerDateTime
IsUnread
IsCorresp
IsUserShortcut
AlternateKey1
= FolderId
+ SortKey
AlternateKey2
= TriggerDateTime
MessageSummary
52
MessageId
FolderExcludeList
IsUnread
IsCorresp
MessageDateTime
DisplayNames
Subject
AttachSummary
53
MessageId
AttachId
AttachType
AttachName
AttachSize

FIG. 8

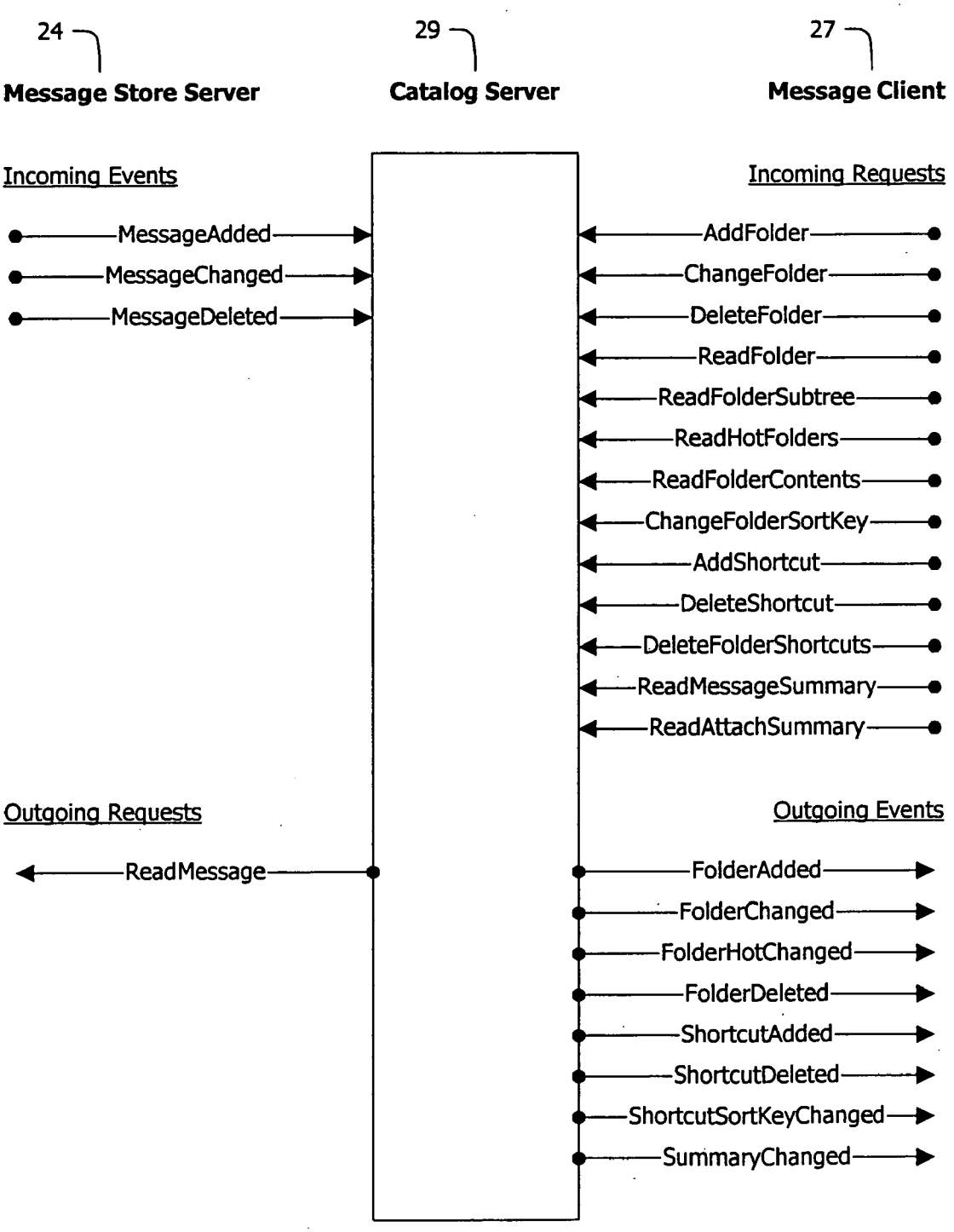

24
Message Store Server
29
Catalog Server
27
Message Client
Incoming Events
MessageAdded
MessageChanged
MessageDeleted
Incoming Requests
AddFolder
ChangeFolder
DeleteFolder
ReadFolder
ReadFolderSubtree
ReadHotFolders
ReadFolderContents
ChangeFolderSortKey
AddShortcut
DeleteShortcut
DeleteFolderShortcuts
ReadMessageSummary
ReadAttachSummary
Outgoing Requests
ReadMessage
Outgoing Events
FolderAdded
FolderChanged
FolderHotChanged
FolderDeleted
ShortcutAdded
ShortcutDeleted
ShortcutSortKeyChanged
SummaryChanged

FIG. 9

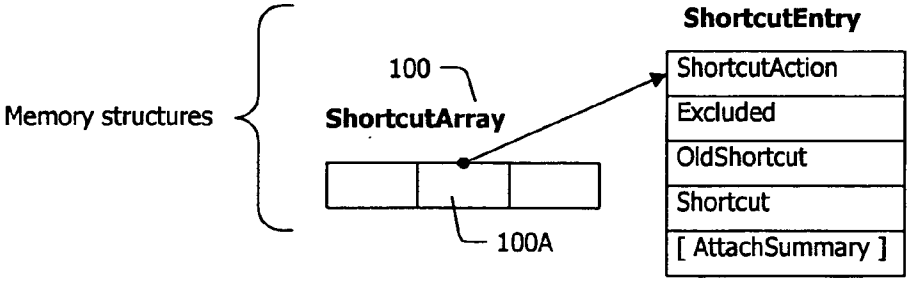


FIG. 10A

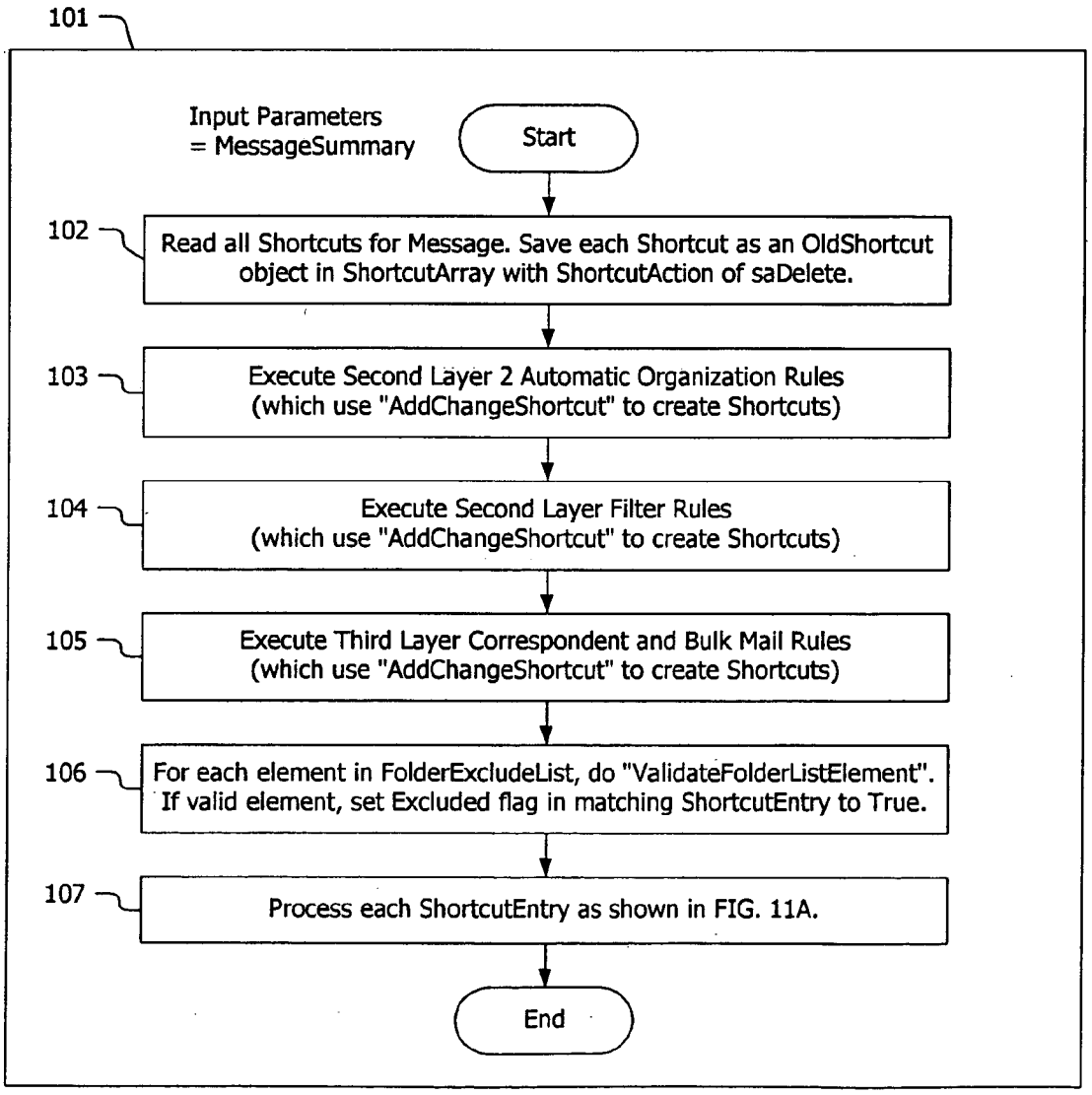


FIG. 10B

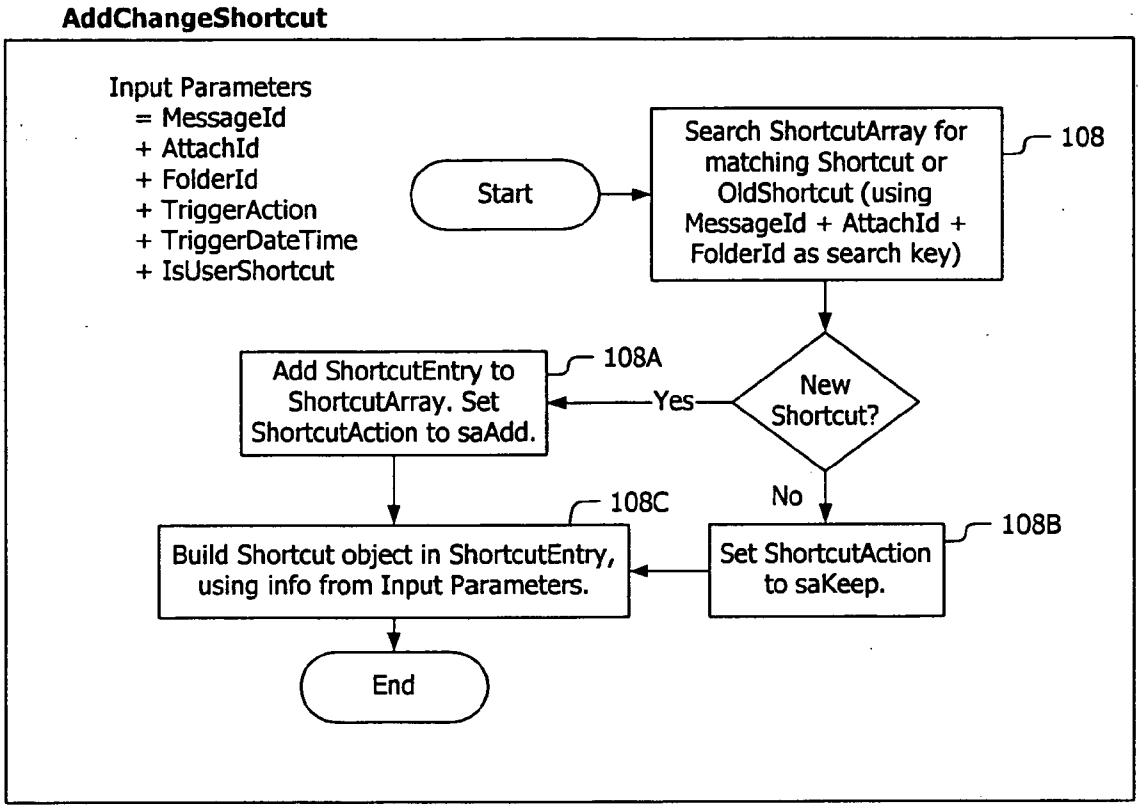


FIG. 10C

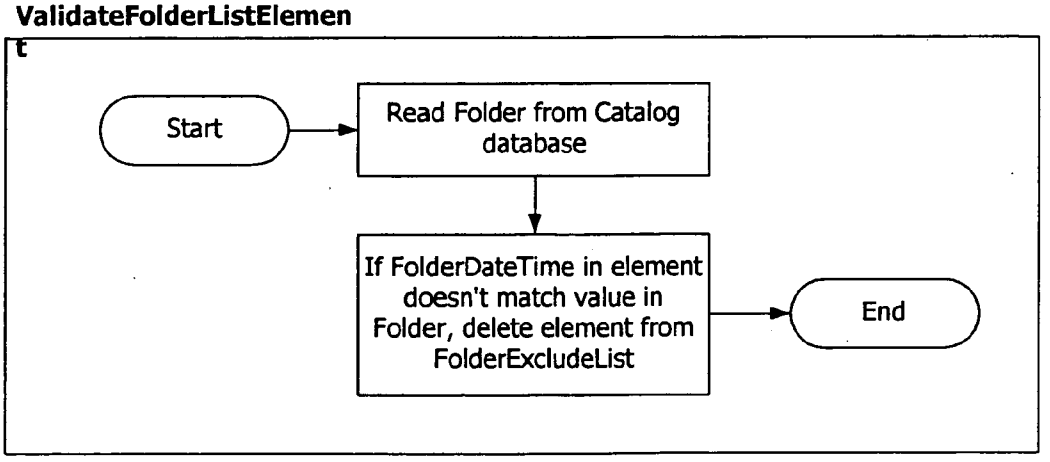


FIG. 10D

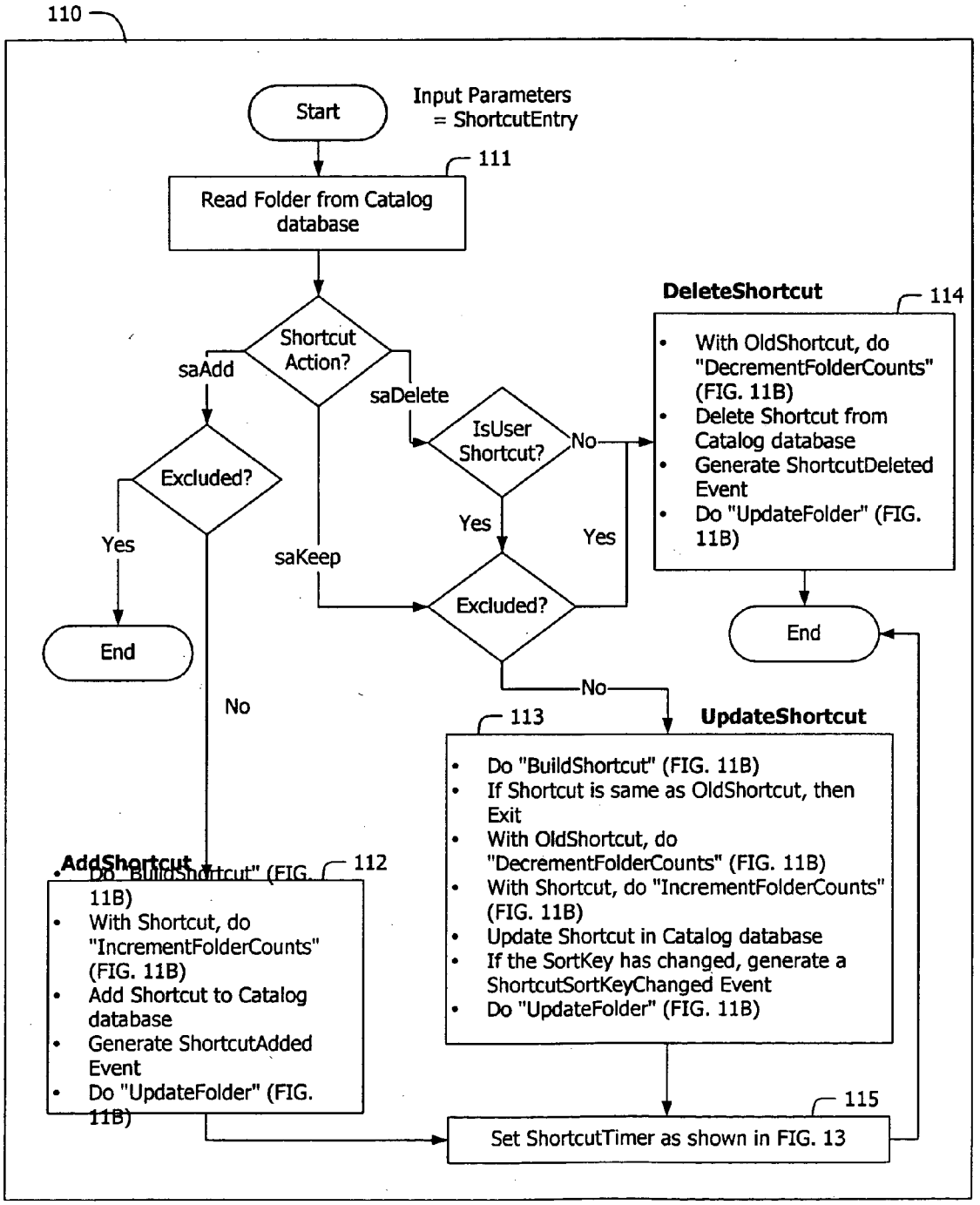

110
Start
Input Parameters = ShortcutEntry
111
Read Folder from Catalog database
Shortcut Action?
saAdd
saDelete
saKeep
Excluded?
Yes
End
No
IsUser Shortcut?
No
Yes
Yes
Excluded?
No
DeleteShortcut
114
• With OldShortcut, do "DecrementFolderCounts" (FIG. 11B)
• Delete Shortcut from Catalog database
• Generate ShortcutDeleted Event
• Do "UpdateFolder" (FIG. 11B)
End
113
UpdateShortcut
• Do "BuildShortcut" (FIG. 11B)
• If Shortcut is same as OldShortcut, then Exit
• With OldShortcut, do "DecrementFolderCounts" (FIG. 11B)
• With Shortcut, do "IncrementFolderCounts" (FIG. 11B)
• Update Shortcut in Catalog database
• If the SortKey has changed, generate a ShortcutSortKeyChanged Event
• Do "UpdateFolder" (FIG. 11B)
AddShortcut
112
• Do "BuildShortcut" (FIG. 11B)
• With Shortcut, do "IncrementFolderCounts" (FIG. 11B)
• Add Shortcut to Catalog database
• Generate ShortcutAdded Event
• Do "UpdateFolder" (FIG. 11B)
115
Set ShortcutTimer as shown in FIG. 13

FIG. 11A

BuildShortcut

Finish building the Shortcut object in ShortcutEntry:
-- Build SortKey is shown in FIG. 12
-- Set SortKey and SortColumn (from Folder)
-- Set IsUnread and IsCorresp (from MessageSummary)

IncrementFolderCounts

- Increment ShortcutCount
- If Shortcut.IsUnread, increment UnreadCount
- If Shortcut.IsCorresp, increment CorrespShortcutCount
- If Shortcut.IsCorresp AND IsUnread, increment UnreadCorrespCount

DecrementFolderCounts

- Decrement ShortcutCount
- If Shortcut.IsUnread, decrement UnreadCount
- If Shortcut.IsCorresp, decrement CorrespShortcutCount
- If Shortcut.IsCorresp AND IsUnread, decrement UnreadCorrespCount

UpdateFolder

- Update Folder in Catalog database
- Generate FolderChanged Event

FIG. 11B

SHORTCUT STRUCTURE

Byte Offset	Length	Field
0	4	MessageId
4	2	AttachId
6	4	FolderId
10	6	SortKey
16	1	SortColumn
17	1	TriggerAction
18	4	TriggerDateTime
22	1	Flags
23	1	IsCorresp

FLAGS ENCODING

Bit	Length	Field
0	1	SortDirection
1	1	IsUnread
2	1	IsUserShortcut

FIELD TYPE	SORT KEY FORMAT	Notes
UnicodeString	UCW1 | UCW2 | UCW3	Unicode Character Weights of first 3 characters of string.
AnsiString	C1 | C2 | C3 | C4 | C5 | C6	Values of first 6 characters of string.
DateTime	YY | MM | DD | hh | mm | ss	YY = Year - 1950
Enumeration OR Boolean (sdDescending)	Value | YY | MM | DD | hh	YY = Year - 1950
Enumeration OR Boolean (sdAscending)	Value | 255 - YY | 255 - MM | 255 - DD | 255 - hh	YY = Year - 1950
Integer32 OR Integer16	---------- Bits ---------- 31-24 | 23-16 | 15-8 | 7-0 | Not Used	Bits are numbered with 0 being the least significant and 31 the most significant.

FIG. 12

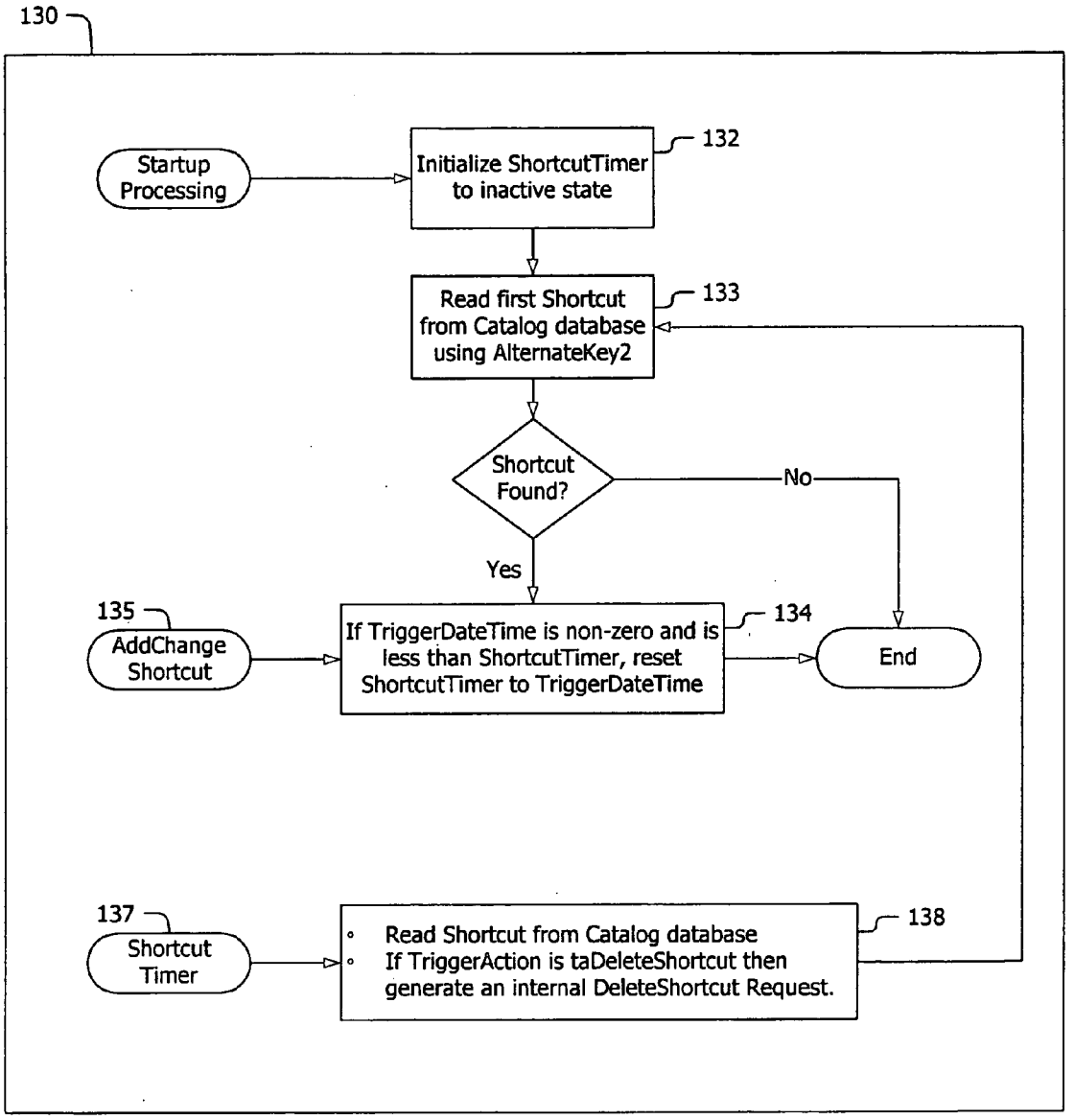

130
Startup Processing
Initialize ShortcutTimer to inactive state
132
Read first Shortcut from Catalog database using AlternateKey2
133
Shortcut Found?
No
Yes
135
AddChange Shortcut
If TriggerDateTime is non-zero and is less than ShortcutTimer, reset ShortcutTimer to TriggerDateTime
134
End
137
Shortcut Timer
Read Shortcut from Catalog database
If TriggerAction is taDeleteShortcut then generate an internal DeleteShortcut Request.
138

FIG. 13

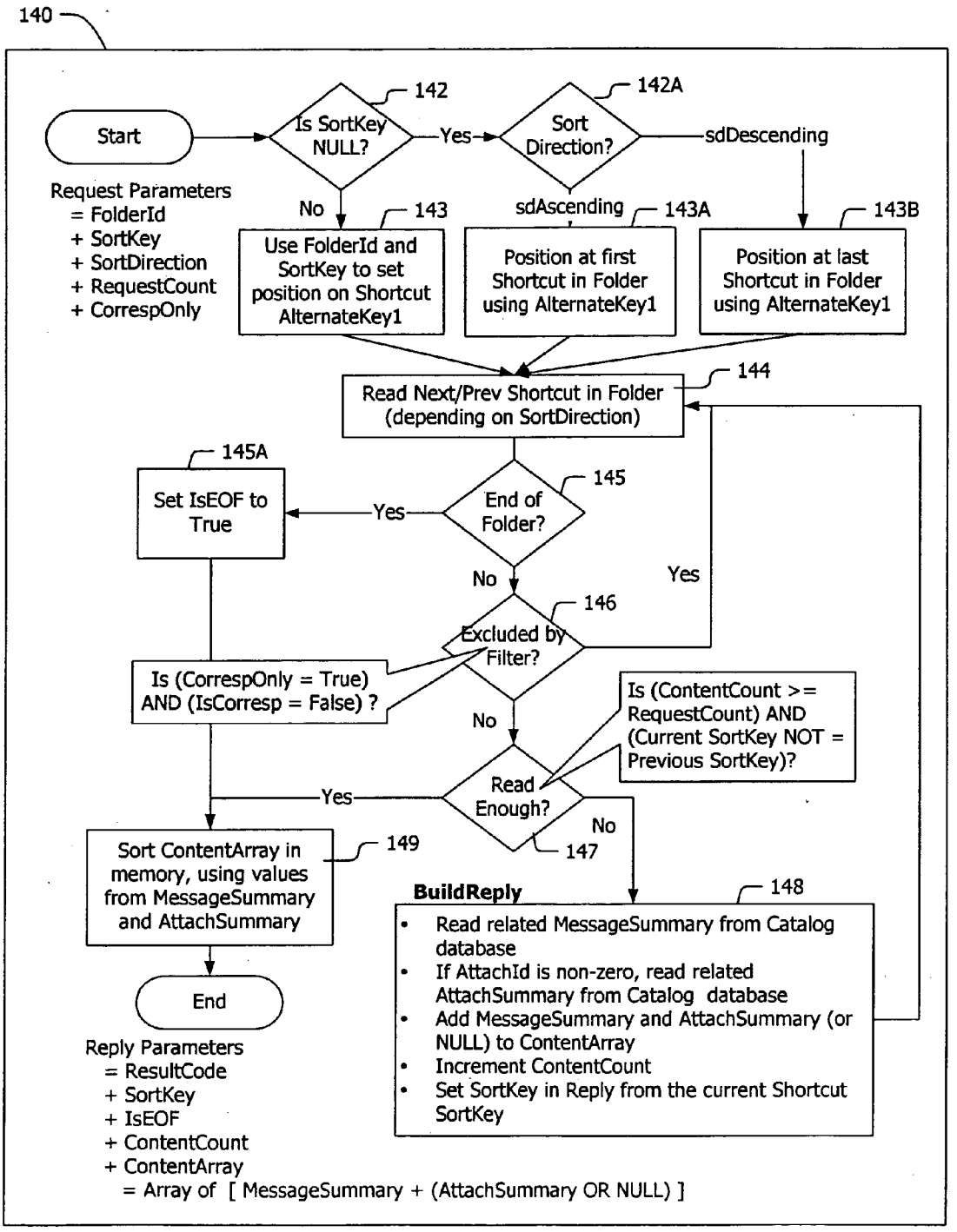

140
Start
142
Is SortKey NULL?
Yes
142A
Sort Direction?
sdDescending
sdAscending
No
143
Use FolderId and SortKey to set position on Shortcut AlternateKey1
143A
Position at first Shortcut in Folder using AlternateKey1
143B
Position at last Shortcut in Folder using AlternateKey1
Request Parameters
= FolderId
+ SortKey
+ SortDirection
+ RequestCount
+ CorrespOnly
144
Read Next/Prev Shortcut in Folder (depending on SortDirection)
145A
Set IsEOF to True
Yes
145
End of Folder?
No
Yes
146
Excluded by Filter?
Is (CorrespOnly = True) AND (IsCorresp = False) ?
No
Is (ContentCount >= RequestCount) AND (Current SortKey NOT = Previous SortKey)?
Yes
Read Enough?
No
147
149
Sort ContentArray in memory, using values from MessageSummary and AttachSummary
BuildReply
148
• Read related MessageSummary from Catalog database
• If AttachId is non-zero, read related AttachSummary from Catalog database
• Add MessageSummary and AttachSummary (or NULL) to ContentArray
• Increment ContentCount
• Set SortKey in Reply from the current Shortcut SortKey
End
Reply Parameters
= ResultCode
+ SortKey
+ IsEOF
+ ContentCount
+ ContentArray
= Array of [MessageSummary + (AttachSummary OR NULL)]

FIG. 14

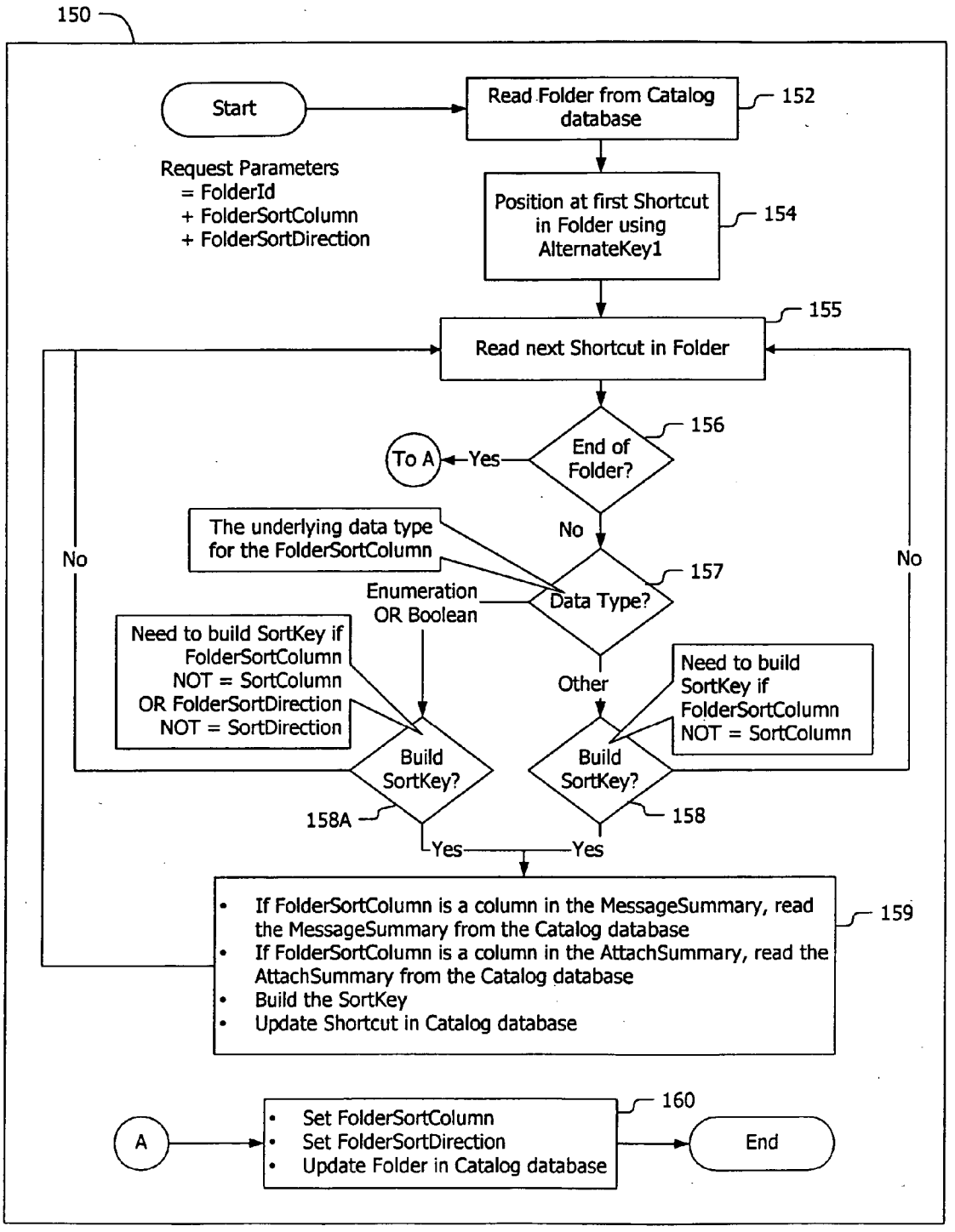

150
Start
Read Folder from Catalog database
152
Request Parameters
= FolderId
+ FolderSortColumn
+ FolderSortDirection
Position at first Shortcut in Folder using AlternateKey1
154
155
Read next Shortcut in Folder
156
End of Folder?
Yes
To A
No
The underlying data type for the FolderSortColumn
157
Data Type?
Enumeration OR Boolean
Other
No
No
Need to build SortKey if FolderSortColumn NOT = SortColumn OR FolderSortDirection NOT = SortDirection
Need to build SortKey if FolderSortColumn NOT = SortColumn
Build SortKey?
158A
Build SortKey?
158
Yes
Yes
If FolderSortColumn is a column in the MessageSummary, read the MessageSummary from the Catalog database
If FolderSortColumn is a column in the AttachSummary, read the AttachSummary from the Catalog database
Build the SortKey
Update Shortcut in Catalog database
159
160
A
Set FolderSortColumn
Set FolderSortDirection
Update Folder in Catalog database
End

FIG. 15

Contains all tables and fields of Base Layer, plus the following:

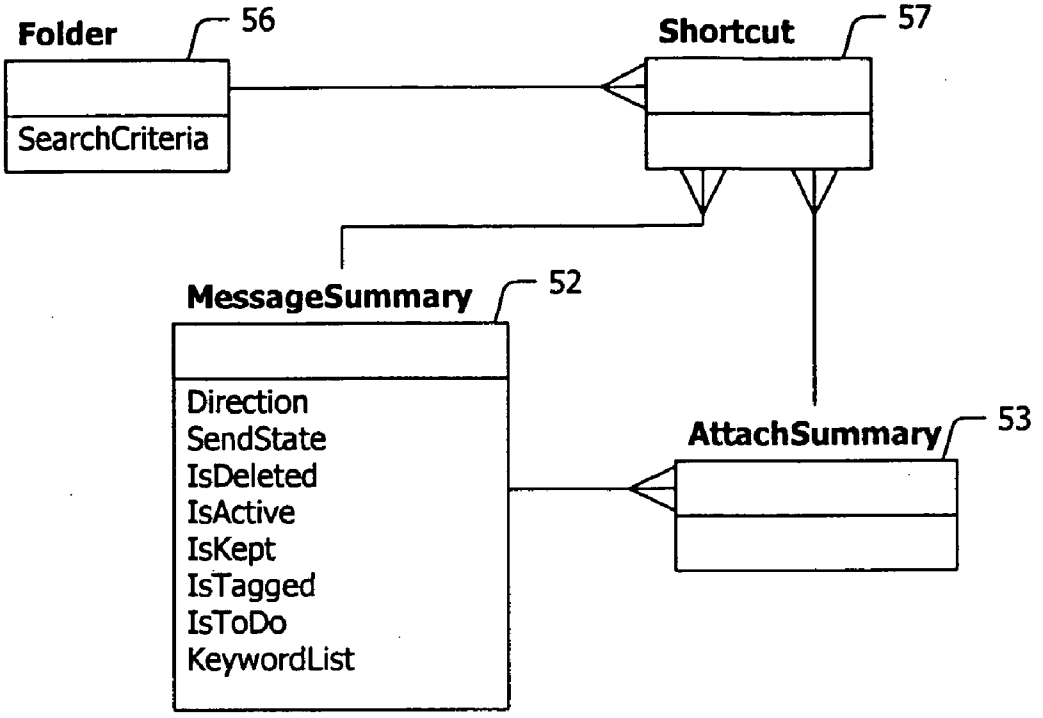


FIG. 16

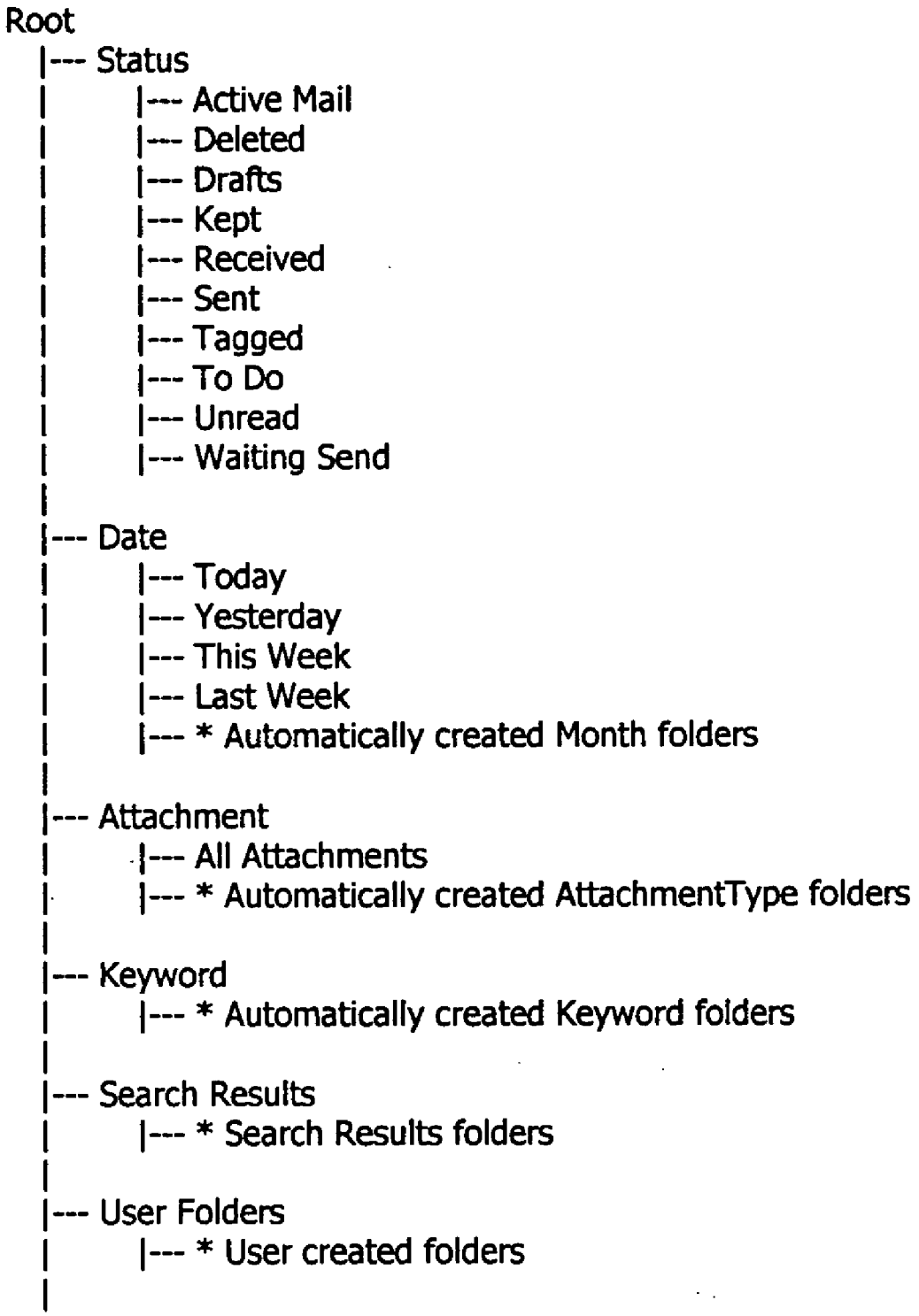
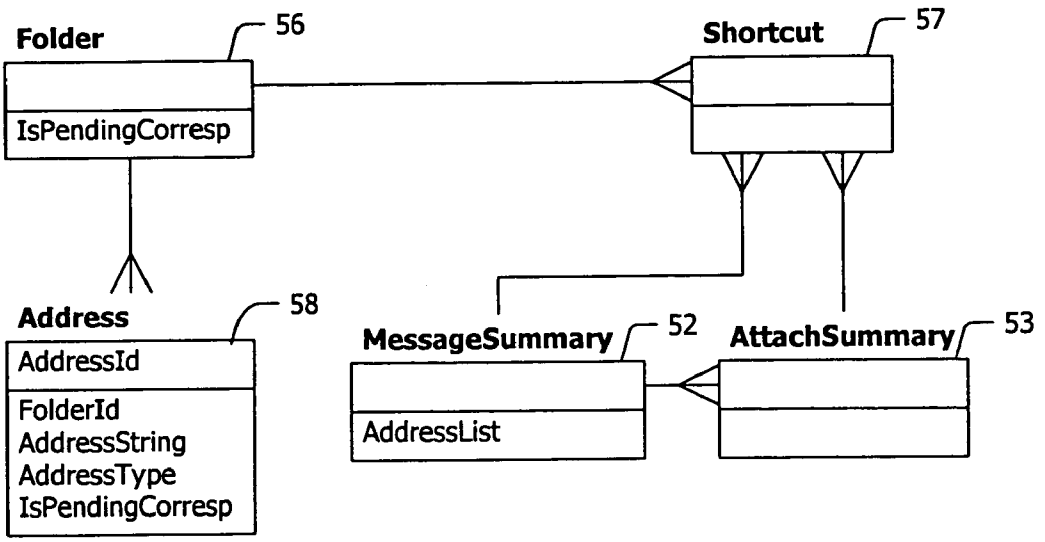

Root
|--- Status
| |--- Active Mail
| |--- Deleted
| |--- Drafts
| |--- Kept
| |--- Received
| |--- Sent
| |--- Tagged
| |--- To Do
| |--- Unread
| |--- Waiting Send
|
|--- Date
| |--- Today
| |--- Yesterday
| |--- This Week
| |--- Last Week
| |--- * Automatically created Month folders
|
|--- Attachment
| |--- All Attachments
| |--- * Automatically created AttachmentType folders
|
|--- Keyword
| |--- * Automatically created Keyword folders
|
|--- Search Results
| |--- * Search Results folders
|
|--- User Folders
| |--- * User created folders
|

FIG. 17

Contains all tables and fields of Base Layer and Second Layer, plus the following:



AlternateKey1
= AddressString
(uppercase)

AlternateKey2
= FolderId

FIG. 18

Contains all Folders in Base Layer and Second Layer, plus the following:

<Root>
|--- Correspondents
| |--- Me
| |--- * Automatically created Correspondent folders
| |--- * User created Correspondent folders
|
|--- Bulk Mail
| |--- Unsorted
| |--- * User created Bulk Mail folders

FIG. 19

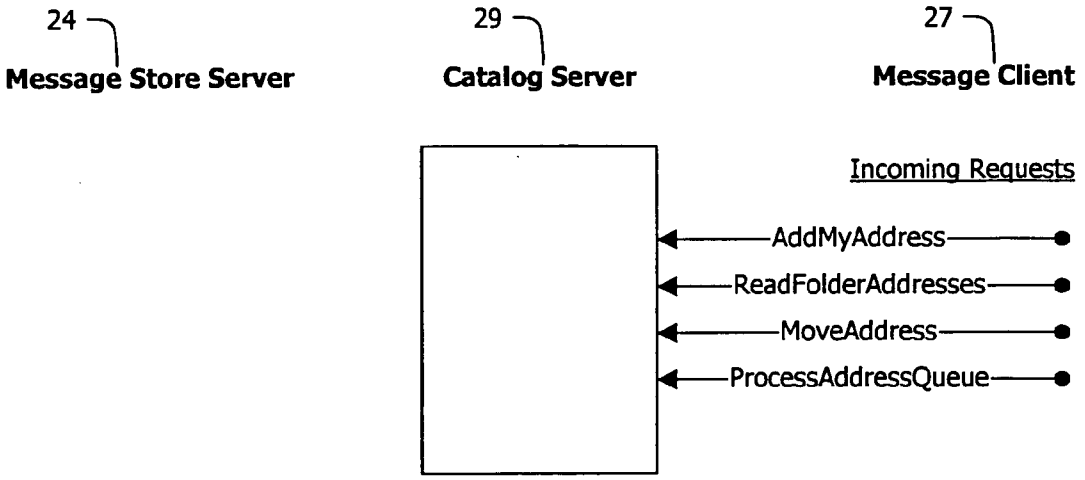

Includes all Requests and Events in Base Layer and Second Layer, plus the following:
24
Message Store Server
29
Catalog Server
27
Message Client
Incoming Requests
AddMyAddress
ReadFolderAddresses
MoveAddress
ProcessAddressQueue

FIG. 20

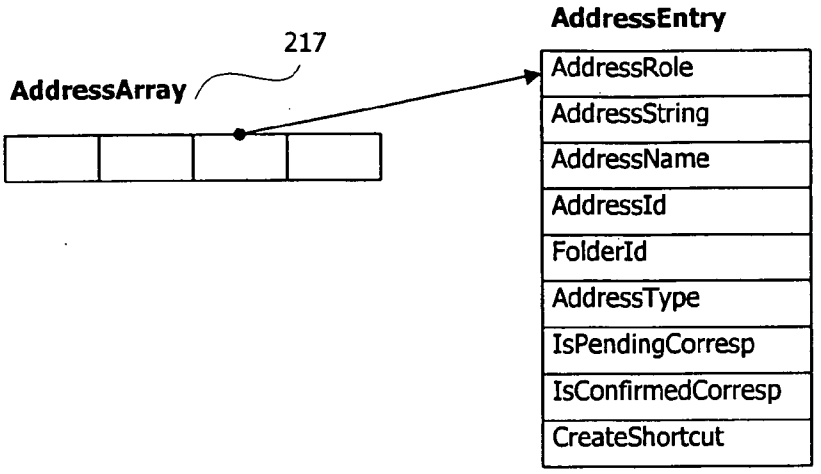


FIG. 21A

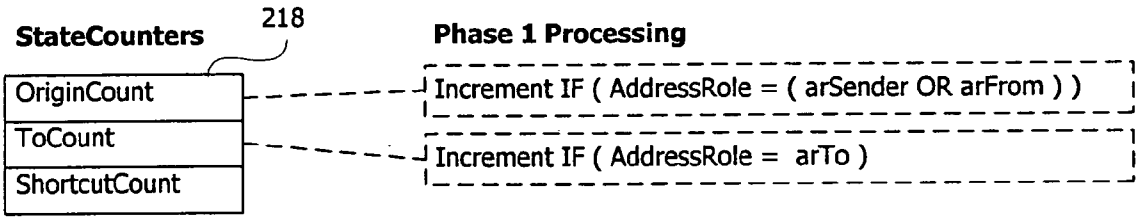


FIG. 21B

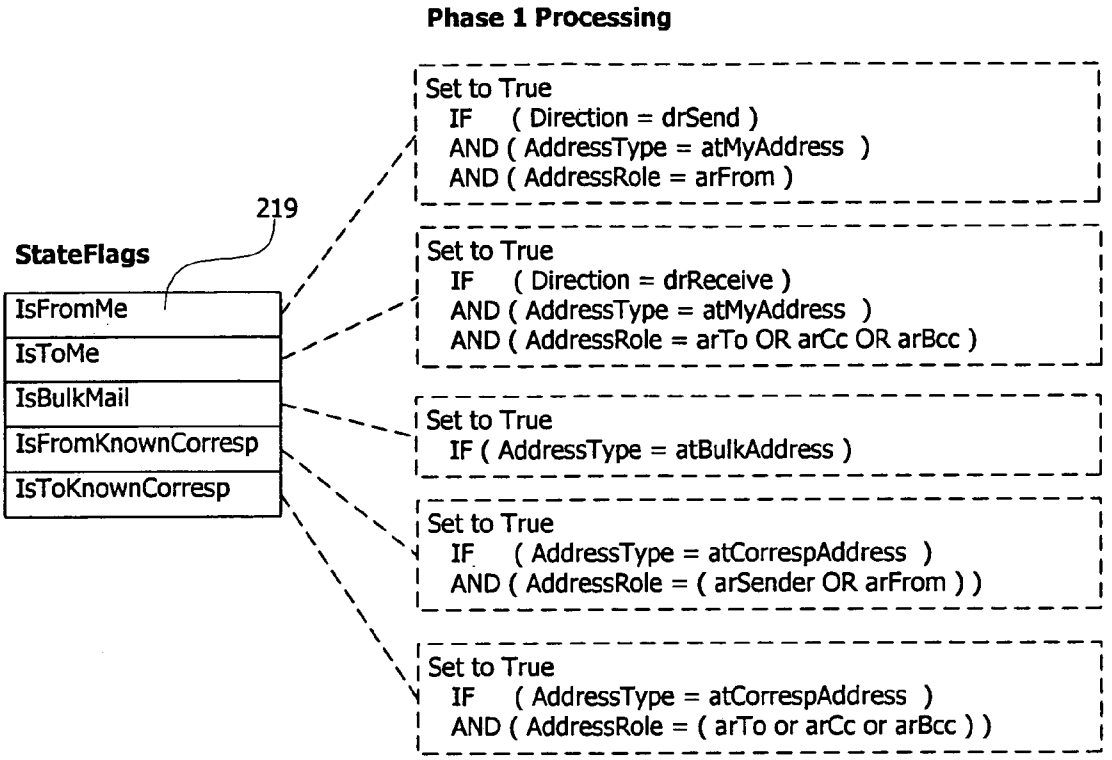

Phase 1 Processing
219
StateFlags
IsFromMe
IsToMe
IsBulkMail
IsFromKnownCorresp
IsToKnownCorresp
Set to True
IF (Direction = drSend)
AND (AddressType = atMyAddress)
AND (AddressRole = arFrom)
Set to True
IF (Direction = drReceive)
AND (AddressType = atMyAddress)
AND (AddressRole = arTo OR arCc OR arBcc)
Set to True
IF (AddressType = atBulkAddress)
Set to True
IF (AddressType = atCorrespAddress)
AND (AddressRole = (arSender OR arFrom))
Set to True
IF (AddressType = atCorrespAddress)
AND (AddressRole = (arTo or arCc or arBcc))

FIG. 21C

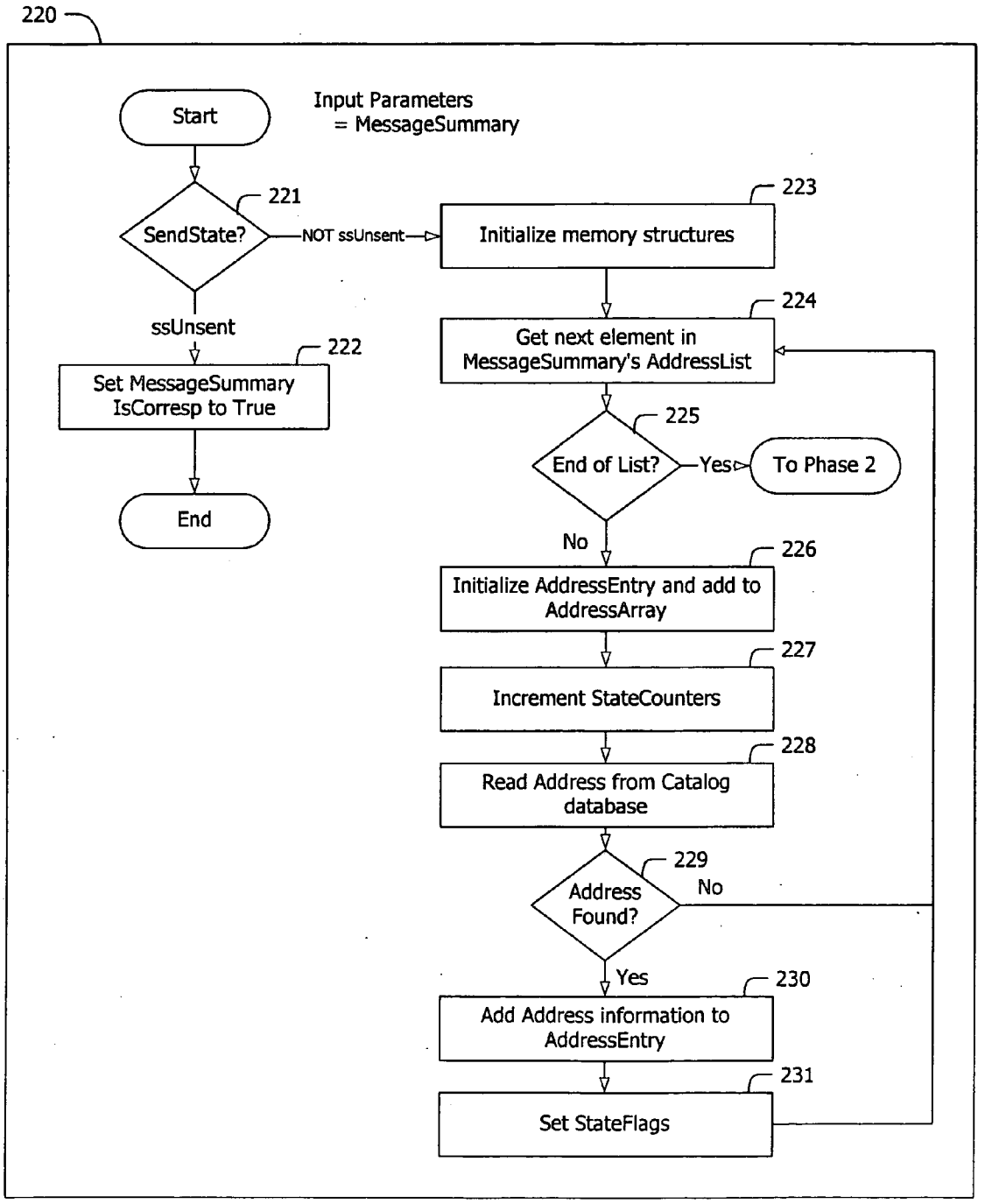

220
Start
Input Parameters
= MessageSummary
221
SendState?
NOT ssUnsent
ssUnsent
222
Set MessageSummary
IsCorresp to True
End
223
Initialize memory structures
224
Get next element in
MessageSummary's AddressList
225
End of List?
Yes
To Phase 2
No
226
Initialize AddressEntry and add to
AddressArray
227
Increment StateCounters
228
Read Address from Catalog
database
229
Address
Found?
No
Yes
230
Add Address information to
AddressEntry
231
Set StateFlags

FIG. 22

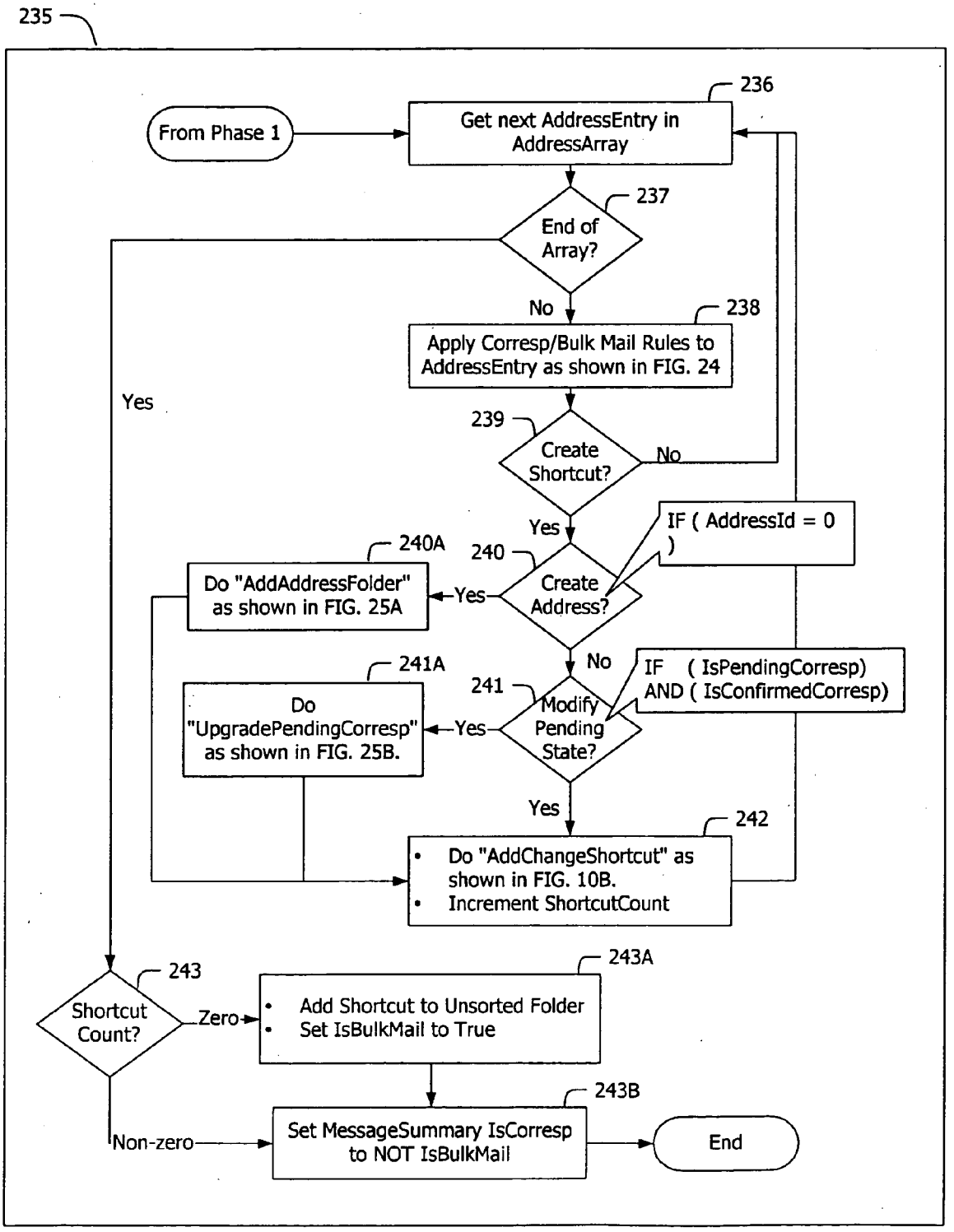

235
From Phase 1
236
Get next AddressEntry in AddressArray
237
End of Array?
No
238
Apply Corresp/Bulk Mail Rules to AddressEntry as shown in FIG. 24
Yes
239
Create Shortcut?
No
Yes
IF (AddressId = 0)
240A
240
Do "AddAddressFolder" as shown in FIG. 25A
Yes
Create Address?
No
IF (IsPendingCorresp) AND (IsConfirmedCorresp)
241A
241
Do "UpgradePendingCorresp" as shown in FIG. 25B.
Yes
Modify Pending State?
Yes
242
• Do "AddChangeShortcut" as shown in FIG. 10B.
• Increment ShortcutCount
243
Shortcut Count?
Zero
243A
• Add Shortcut to Unsorted Folder
• Set IsBulkMail to True
Non-zero
243B
Set MessageSummary IsCorresp to NOT IsBulkMail
End

FIG. 23

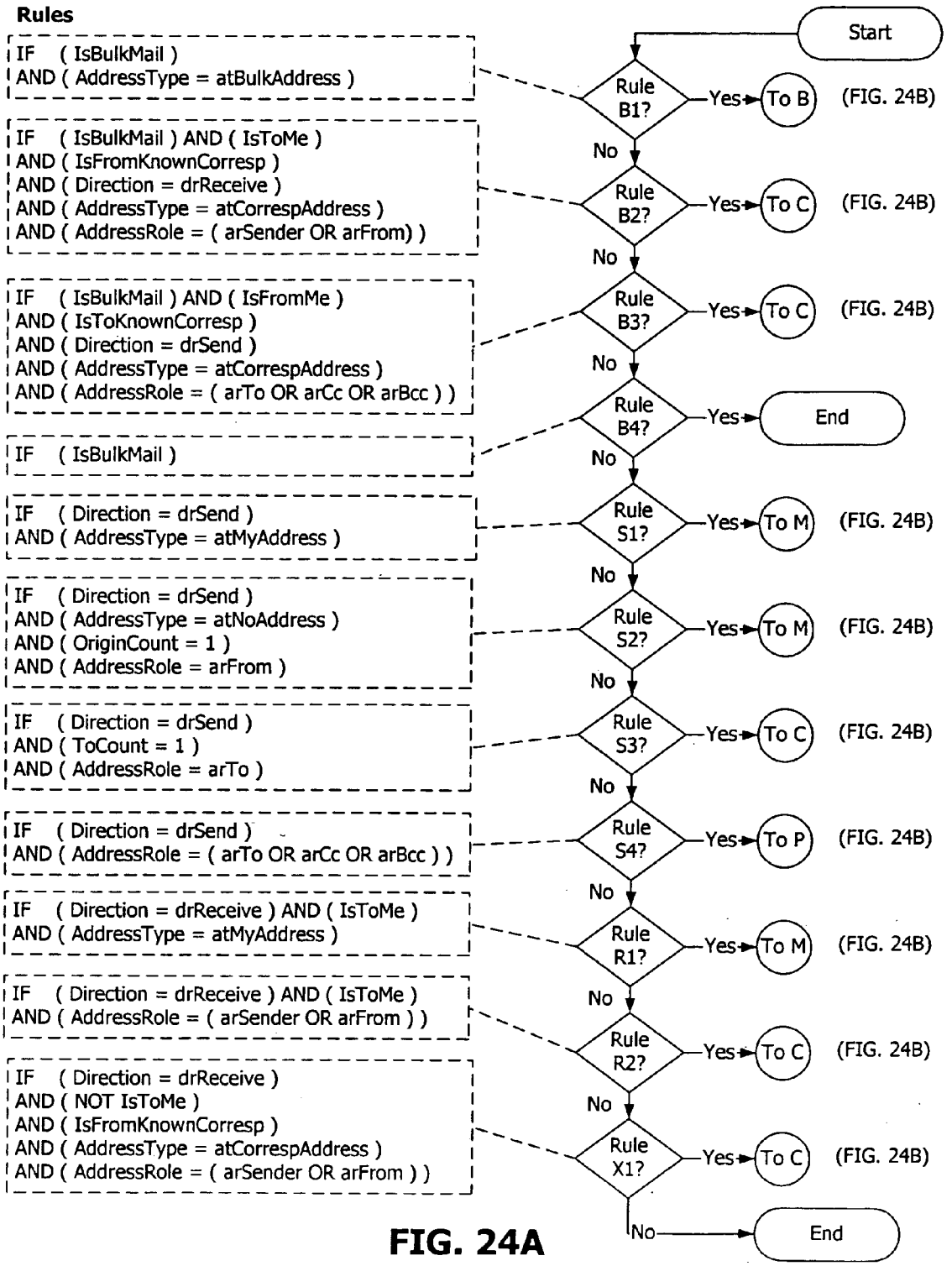

Rules
IF (IsBulkMail)
AND (AddressType = atBulkAddress)
IF (IsBulkMail) AND (IsToMe)
AND (IsFromKnownCorresp)
AND (Direction = drReceive)
AND (AddressType = atCorrespAddress)
AND (AddressRole = (arSender OR arFrom))
IF (IsBulkMail) AND (IsFromMe)
AND (IsToKnownCorresp)
AND (Direction = drSend)
AND (AddressType = atCorrespAddress)
AND (AddressRole = (arTo OR arCc OR arBcc))
IF (IsBulkMail)
IF (Direction = drSend)
AND (AddressType = atMyAddress)
IF (Direction = drSend)
AND (AddressType = atNoAddress)
AND (OriginCount = 1)
AND (AddressRole = arFrom)
IF (Direction = drSend)
AND (ToCount = 1)
AND (AddressRole = arTo)
IF (Direction = drSend)
AND (AddressRole = (arTo OR arCc OR arBcc))
IF (Direction = drReceive) AND (IsToMe)
AND (AddressType = atMyAddress)
IF (Direction = drReceive) AND (IsToMe)
AND (AddressRole = (arSender OR arFrom))
IF (Direction = drReceive)
AND (NOT IsToMe)
AND (IsFromKnownCorresp)
AND (AddressType = atCorrespAddress)
AND (AddressRole = (arSender OR arFrom))
Start
Rule B1?
Yes
To B
(FIG. 24B)
No
Rule B2?
Yes
To C
(FIG. 24B)
No
Rule B3?
Yes
To C
(FIG. 24B)
No
Rule B4?
Yes
End
No
Rule S1?
Yes
To M
(FIG. 24B)
No
Rule S2?
Yes
To M
(FIG. 24B)
No
Rule S3?
Yes
To C
(FIG. 24B)
No
Rule S4?
Yes
To P
(FIG. 24B)
No
Rule R1?
Yes
To M
(FIG. 24B)
No
Rule R2?
Yes
To C
(FIG. 24B)
No
Rule X1?
Yes
To C
(FIG. 24B)
No
End

FIG. 24A

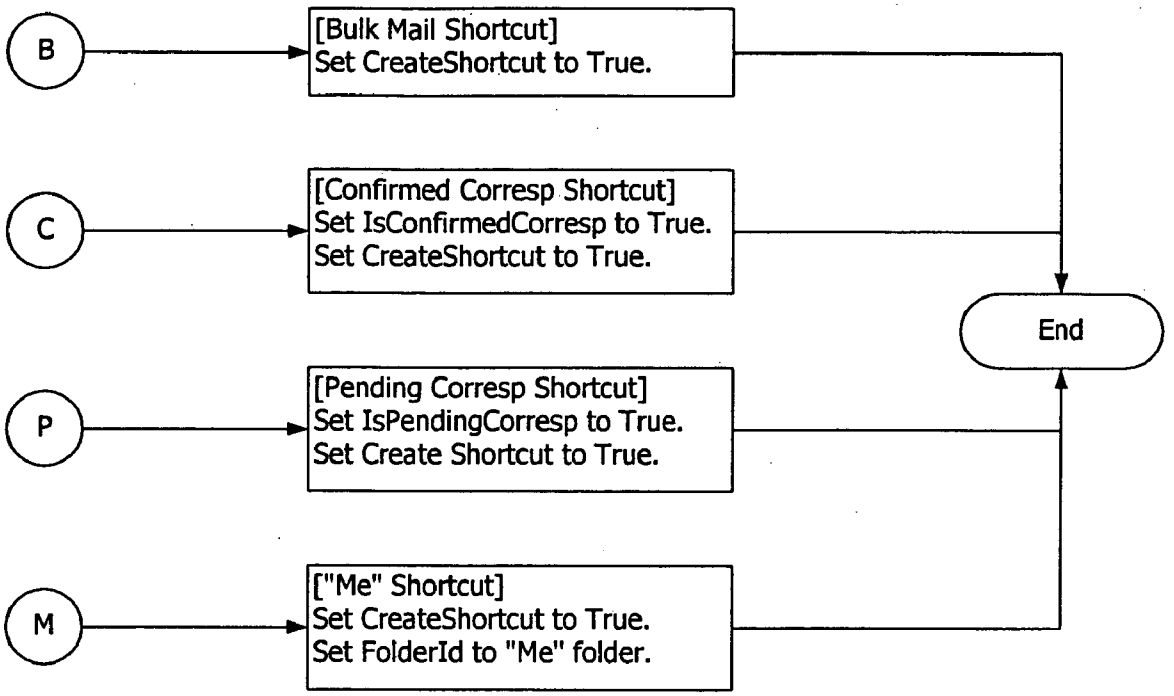

B
[Bulk Mail Shortcut]
Set CreateShortcut to True.
C
[Confirmed Corresp Shortcut]
Set IsConfirmedCorresp to True.
Set CreateShortcut to True.
End
P
[Pending Corresp Shortcut]
Set IsPendingCorresp to True.
Set Create Shortcut to True.
M
["Me" Shortcut]
Set CreateShortcut to True.
Set FolderId to "Me" folder.

FIG. 24B

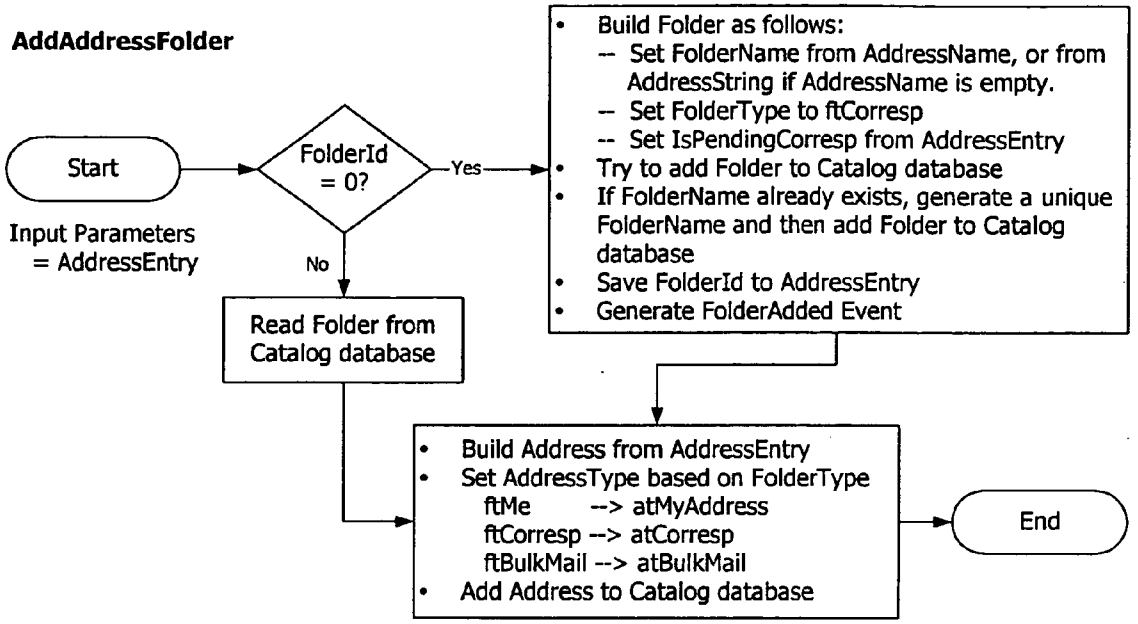


FIG. 25A

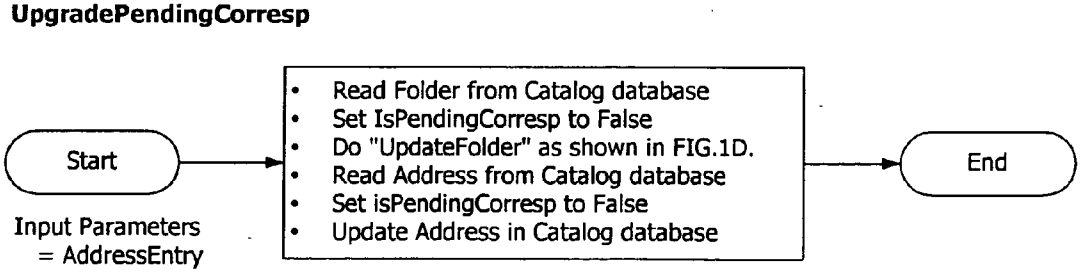


FIG. 25B

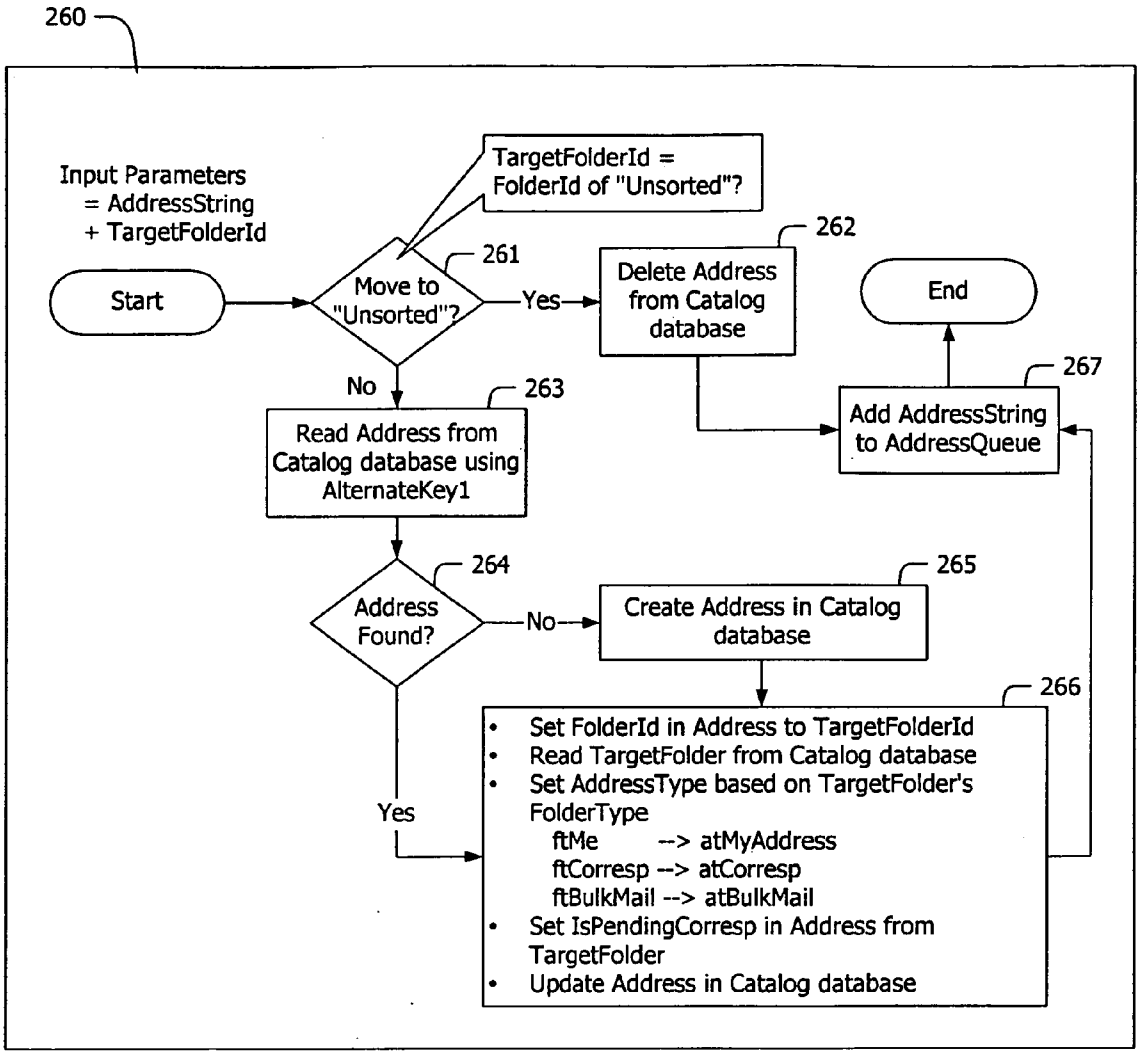

260
Input Parameters
= AddressString
+ TargetFolderId
TargetFolderId = FolderId of "Unsorted"?
Start
Move to "Unsorted"?
261
Yes
Delete Address from Catalog database
262
End
No
263
Read Address from Catalog database using AlternateKey1
Add AddressString to AddressQueue
267
Address Found?
264
No
Create Address in Catalog database
265
Yes
266
• Set FolderId in Address to TargetFolderId
• Read TargetFolder from Catalog database
• Set AddressType based on TargetFolder's FolderType
ftMe --> atMyAddress
ftCorresp --> atCorresp
ftBulkMail --> atBulkMail
• Set IsPendingCorresp in Address from TargetFolder
• Update Address in Catalog database

FIG. 26

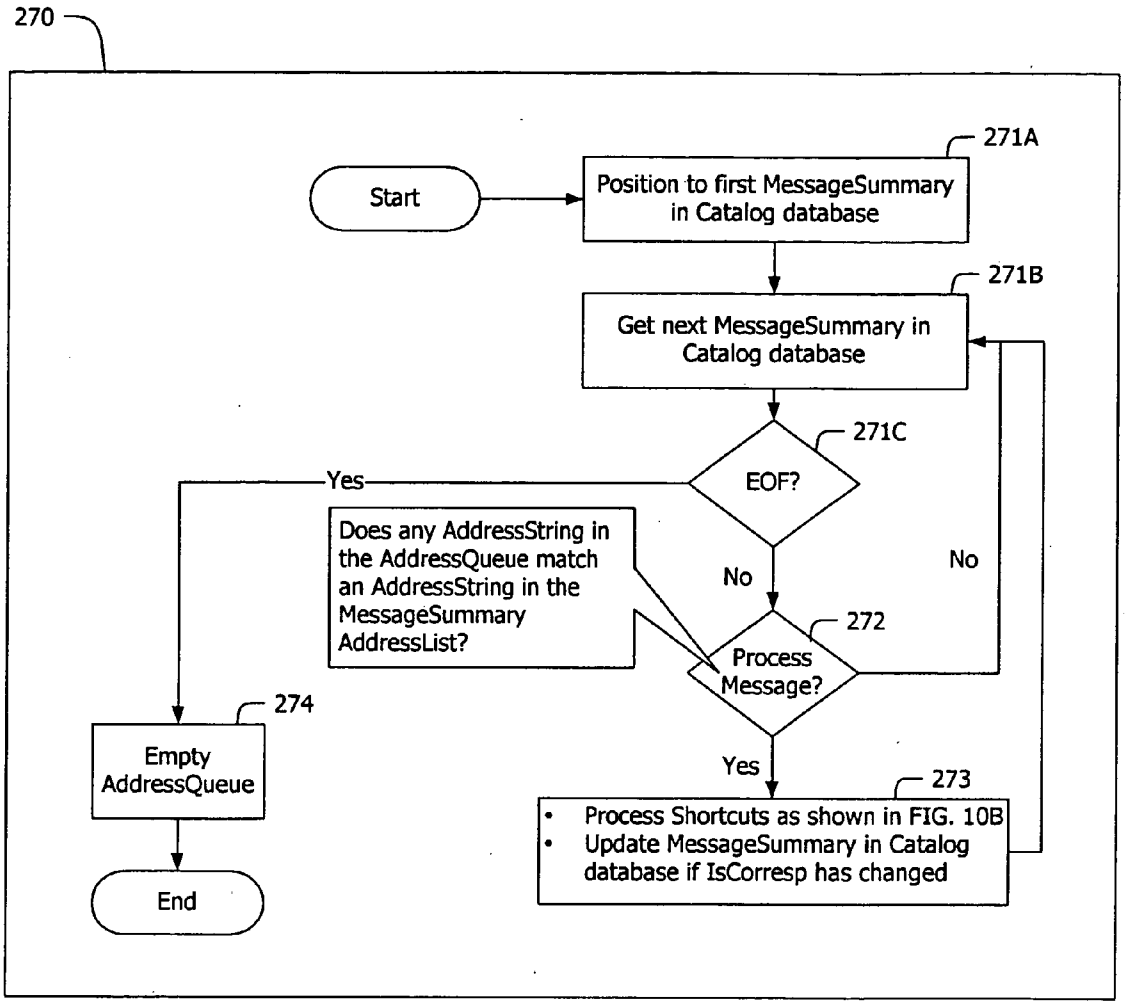

270
271A
Start
Position to first MessageSummary in Catalog database
271B
Get next MessageSummary in Catalog database
271C
EOF?
Yes
No
Does any AddressString in the AddressQueue match an AddressString in the MessageSummary AddressList?
No
272
Process Message?
Yes
273
• Process Shortcuts as shown in FIG. 10B
• Update MessageSummary in Catalog database if IsCorresp has changed
274
Empty AddressQueue
End

FIG. 27

SYSTEM FOR MANAGING AND ORGANIZING STORED ELECTRONIC MESSAGES

REFERENCE TO RELATED APPLICATION

[0001] This is a continuation of application Ser. No. 09/704,199 filed on 30 Oct. 2000 and entitled System for Managing and Organizing Stored Electronic Messages.

FIELD OF THE INVENTION

[0002] This invention relates to electronic messaging systems and, in particular relates to systems for managing and organizing electronic messages. Messages may be e-mail messages, voice mail messages, digitized faxes or the like. Specific aspects of the invention provide computer-implemented methods for managing and organizing electronic messages, computer systems for managing and organizing electronic messages, and computer-readable media containing computer instructions which, when executed by the computer cause the computer to perform a method according to the invention.

BACKGROUND OF THE INVENTION

[0003] Electronic messaging, which includes electronic mail (or "e-mail") messaging, is now an accepted, and some would say vital, medium for business and personal communications. The rapid growth of electronic messaging is expected to continue. This growth brings an increasingly serious problem of how to manage the volume of messages. According to a 1998 Pitney Bowes survey, 71% of respondents said they felt overwhelmed by the number of messages they receive. This problem is becoming more severe. John Dvorak, a frequent writer on the topic of computing states in PC Computing magazine that ". . . we have poor tools to sort and organize (or even find) the e-mail we collect".

[0004] Electronic messages, which may include attachments of diverse kinds, are sent and received through the use of messaging software. For example, e-mail messages are sent and received by e-mail software such as Microsoft's OUTLOOK™ or Netscape's COMMUNICATOR™. Other widely used types of electronic messaging are voice mail, fax and instant messaging. The vast majority of current messaging software is based on design principles that originated when message volumes were low. Current e-mail software, for example, provides rudimentary features for organizing e-mail messages (both incoming and outgoing) into various folders. The most basic model saves received messages in an Inbox folder, messages waiting -delivery in an Outbox folder, and sent messages in a Sent Messages folder. Users can create additional user folders to which they can move or copy messages. Refinements to this basic model include providing additional system folders such as Drafts and Wastebasket folders. In general, the user is responsible for moving e-mail messages between folders and for managing the messages once they have been placed into a folder. This can be an onerous responsibility, especially in cases where the user receives large volumes of e-mail messages as may easily occur, for example, if the user subscribes to one or more high volume mailing lists.

[0005] A fundamental weakness of this folder/message model is that a message can only exist in a single folder at a time. While a user can place copies of e-mail messages into multiple folders the user must manage the copies separately. If the user wishes to see a message in multiple folders, then he or she must make multiple copies of the message, which results in using additional storage space and in creating more messages that need to be managed. This model also requires that the user manually organize each message. This can lead to cluttered folders and a general lack of organization in the stored e-mail messages that a user accumulates over time.

[0006] Further, once an e-mail message has been received it can be difficult to find the message later, especially if there are many folders into which the message could have been placed. This is frustrating and inefficient for the user.

[0007] Some current electronic mail software is capable of filtering incoming e-mail messages by applying a series of rules. The filtering rules may be automatically executed each time a message is sent or received. The current version of Microsoft Outlook has a facility which allows users to create such rules, for example. U.S. Pat. No. 6,057,841, Thurlow et al., describes a system for applying a set of electronic message processing rules for managing incoming and outgoing electronic messages. U.S. Pat. No. 5,377,354, Scannell et al., also describes a rules-based filtering mechanism.

[0008] Rules can execute specific tasks when user-defined criteria are met. Rules can be used to process electronic messages without requiring users to spent a lot of time sorting through their inboxes deleting, filing, and responding to their messages. While filter rules are powerful, they are also difficult to use because they are typically implemented as a series of instructions against which each message is evaluated. If the number of rules exceeds a relatively small number the overall rule set becomes very difficult to understand. Another disadvantage is that rules must be created manually and can involve a significant amount of effort if a user wants to organize their messages in a thorough manner, such as by correspondent. A further disadvantage is that e-mail systems which apply filtering rules are typically restricted by the folder/message model and cannot organize a message into multiple folders without creating multiple copies of the message. As a result of the foregoing disadvantages many users do not bother to set up such rules. Even when the rules have been set up they act only when a message is sent or received. Such rules are incapable of managing messages after they have been received or sent.

[0009] Other features which software vendors have provided in an attempt to help users organize their messages are keywords (also referred to as "categories"), tags (also referred to as "flags"), searching tools, and links to other objects such as task lists. While these features improve the manageability of e-mail they are less powerful than filtering rules and have proven inadequate for dealing with higher message volumes.

[0010] Keywords and tags let a user highlight and identify messages to distinguish them from other stored messages. A major drawback to these mechanisms is that the highlighted messages are visible only in the folder to which the message belongs. The value of these mechanisms is significantly reduced because there is no fast and convenient way to locate all tagged messages or all messages that have been assigned a given keyword.

[0011] Searching is an important tool in dealing with large message volumes. Traditional sequential search techniques are usually too time-consuming to make them very useful

for larger message stores. As a result, there have been recent efforts to provide systems which implement full-text indexing and retrieval capabilities for message stores. While searching is an important technique for finding previously sent or received messages, it is not particularly useful or efficient for dealing with messages as they are received and then handled by the user. A search must be performed each time a user wishes to access messages which match a particular set of search criteria. The user is generally forced to manually enter the search criteria. Once a search has been run the results of the search may be placed into a separate "search results" folder (in addition to the folder in which the original message resides). Search results folders are not generally useful for organizing electronic messages because of their limited capabilities. Such folders cannot form the basis for a more general purpose solution for organizing messages into multiple folders. For example the Microsoft Platform SDK describes some limitations of MAPI search results folders as follows:

[0012] The only way that the contents of a search-results folder can be modified is through the IMAPIContainer::SetSearchCriteria call;

[0013] Messages cannot be moved or copied into or out of search-results folders; and,

[0014] Search-results folders cannot contain subfolders.

[0015] Some software packages allow objects to be linked to one another. For example, Microsoft Outlook 2000 has a task list. A user can add a shortcut to an electronic message to a task. Outlook 2000 does not provide any facilities to act on the message to which the shortcut refers. The Outlook 2000 task list is not flexible enough for use in the effective organization of electronic messages.

[0016] A number of attempts have been made to overcome problems associated with the current folder/message model. These include U.S. Pat. No. 5,948,058, Kudoh et al., which describes a method for cataloging a message into multiple categories. An array of bitmaps in the message is used to identify the categories to which a message belongs. This is a very simple implementation that would have serious performance problems when searching for all messages that belong to a category—all messages in the message store would need to be examined.

[0017] U.S. Pat. No. 6,029,164, Birrell et al., describes a method for adding labels to messages which are then indexed by a full-text index and retrieval engine. An advantage of the Birrell et al. system over the system of Kudoh et al. is that it provides a rapid global search capability for finding messages with the desired label, and provides the ability for the user to add, modify or delete labels. Some disadvantages of the Birrell et al. system are that messages are processed in batches (for performance reasons) so the index is not always current. Further, a user must execute a search to find messages associated with the desired label.

[0018] Miller et al, PCT patent publication No. WO99/04344 disclose an e-mail system in which messages can be accessed in a correspondent-centric manner. The Miller et al. system uses a relational database to organize messages. The underlaying database structure is conventional in design and has a Correspondent table, a Message table and a Message-Correspondent Relationship table. While Miller et al. do provide a system which addresses some of the problems addressed above, Miller et al. do not provide a general solution to the problem of organizing a store of electronic messages. Some shortcomings of the Miller et al. system are as follows:

[0019] a separate set of tables is required for each field by which messages are to be organized. The system disclosed by Miller et al. only permits messages to be organized by topic and correspondent;

[0020] messages are organized by e-mail address—no support is provided for grouping messages by correspondent where the correspondent has multiple e-mail addresses;

[0021] the list of correspondents can become cluttered with persons with whom no real correspondence is conducted. For example, a large number of correspondent entries would be created if the user accidentally performs a "Reply All" operation on a message that has a large distribution list of unknown correspondents.

[0022] bulk mail is processed in an identical manner to personally addressed mail.

[0023] messages can be read from the database only in ascending MessageId sequence.

[0024] There exist computerized systems for organizing data files of various kinds. However, these systems are, in general, not well adapted for use in organizing electronic messages. Some examples of file management systems are described in U.S. Pat. No. 5,544,360, Lewak et al.; U.S. Pat. No. 5,899,995, Millier et al.; and, U.S. Pat. No. 6,009,442 Chen et al.

[0025] There is a need for systems and methods which can automatically organize stored electronic messages, such as e-mail messages, instant messages, voice messages and fax messages. There is a particular need for such systems and methods which include tools for automatically managing stored messages. Any such system must be fast. Users are generally unwilling to wait for a messaging system to generate a list of electronic messages in a particular folder. There is particular need for systems which allow a user to quickly locate a message or group of messages of interest especially given the ever increasing load of messages that many users have to deal with. Ideally such systems should help users to separate important messages from less important messages and to manage the flow of messages.

SUMMARY OF THE INVENTION

[0026] This invention provides a computer-based system for cataloging, retrieving and manipulating electronic messages saved in a message store. Preferred embodiments of the system may be used to automatically organize each of a large number of saved message into multiple folders based on the contents and attributes of the message as well as to facilitate the manual organization of messages. Apparatus according to the invention provides a relational database which uses lightweight message shortcuts to make individual messages available in multiple folders simultaneously. The invention can advantageously be integrated with messaging client software and/or messaging server

software, such as e-mail software, to facilitate the organization of electronic messages.

[0027] One aspect of the invention provides a method for organizing electronic messages comprising: providing an electronic message, the electronic message comprising a plurality of properties; and, generating a plurality of shortcuts to the electronic message, each of the shortcuts comprising a record in a database, each record comprising a FolderId value associating the shortcut with a different one of a plurality of folders and a MessageId value associating the shortcut with the message.

[0028] Further features and advantages of the invention are described below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0029] In drawings which illustrate non-limiting embodiments of the invention,

[0030] **FIG. 1A** is a schematic view of an example computer network which includes a user computer on which electronic messages are received and stored;

[0031] **FIG. 1B** is a schematic overview of a system for organizing messages according to the invention;

[0032] **FIG. 2** is a diagram that shows schematically the logical software, storage and user interface components in apparatus according to a currently preferred embodiment of the invention;

[0033] **FIG. 3** is an example of how a specific e-mail message may be organized into multiple folders in a system according to the invention;

[0034] **FIGS. 4A, 4B** and 4C are diagrams showing three possible physical configurations for systems according to the invention;

[0035] **FIG. 5A** is an entity-relationship diagram which illustrates a catalog and a message store implemented as separate databases;

[0036] **FIG. 5B** is an entity-relationship diagram which illustrates a catalog and a message store implemented in a single combined database;

[0037] **FIG. 6** is an example of a screen display for a user interface that shows folders and messages in multiple views;

[0038] **FIG. 7** schematically illustrates three functional layers of a preferred embodiment of the invention;

[0039] **FIG. 8** is an entity-relationship diagram of a catalog database, showing elements capable of supporting a base layer of functionality according to the invention;

[0040] **FIG. 9** is a diagram illustrating requests and events generated or handled by a catalog server in providing the base layer of functionality in a preferred embodiment of 7 the invention;

[0041] **FIGS. 10A through 10D** show how all shortcuts for a message can be added, changed or deleted in response to a MessageAdded or a MessageChanged event from the message store server, and show memory structures used in this processing;

[0042] **FIGS. 11A and 11B** together provide a flowchart that shows how an individual shortcut can be added, changed or deleted in a catalog database;

[0043] **FIG. 12** discloses the structure of a shortcut, including details of the structure of the SortKey;

[0044] **FIG. 13** is a flowchart that shows processing in a method that can be used to handle timed shortcuts;

[0045] **FIG. 14** is a flowchart that shows a method that can be used for incrementally reading shortcuts for display to a user;

[0046] **FIG. 15** is a flowchart that shows a method that can be used to change the SortKey for a folder and for all the shortcuts in the folder;

[0047] **FIG. 16** is an entity-relationship diagram of the catalog database of **FIG. 8**, showing additional elements that can be used to support a second layer of functionality;

[0048] **FIG. 17** depicts a possible folder-tree structure, showing folders which can be used in providing the second layer of functionality;

[0049] **FIG. 18** is an entity-relationship diagram of the catalog database of **FIGS. 8 and 16**, showing additional elements that can be used to support a third layer of functionality;

[0050] **FIG. 19** depicts a possible folder-tree structure, showing additional folders that may be used in providing the third layer of functionality;

[0051] **FIG. 20** is a diagram of catalog server requests and events, showing incoming requests which can be provided to support third layer functions;

[0052] **FIGS. 21A, 21B** and 21C show memory structures which can be used for correspondent and bulk mail organization;

[0053] **FIG. 22** is a flowchart that shows the first phase of correspondent and bulk mail organization;

[0054] **FIG. 23** is a flowchart that shows the second phase of correspondent and bulk mail organization;

[0055] **FIGS. 24A and 24B** together provide a flowchart that shows rules that are applied to each AddressEntry by the second phase of correspondent and bulk mail organization;

[0056] **FIGS. 25A and 25B** are flowcharts that show how addresses and folders can be added or updated during the second phase of correspondent and bulk mail organization.

[0057] **FIG. 26** is a flowchart that shows MoveAddress request processing; and, **FIG. 27** is a flowchart that shows ProcessAddressQueue request processing.

List of Reference Numerals

11	data store
12	server computer
13	processor
14	network
15	user interface
16	user computer
17	interface device
18	user computer

-continued

List of Reference Numerals	
19	shortcut
20	system
22	electronic message
23	message store
24	message store server
25	user interface device
26	message transport server
27	message client
28	catalog database
29	catalog server
32	envelope
33	transport header
34	status and organizational information
36	message contents
36A	attachments
38	folder
40A, 40B, 40C	system
50	storage object
51	StoreLink table
51A	StoreId
52A	StoreMessageId
52B	StoreLinkId foreign key
52, 52'	MessageSummary table
53, 53'	AttachSummary table
53A	StoreAttachId foreign key
54, 54'	Message table
55, 55'	Attach table
56	Folder table
57	Shortcut table
58	Address table
60	display
61	folder panel
61A	representation of folders
62	tabbed dialog
62A	date tab
62B	correspondents tab
62C	hot tab
64	messages panel
65	column header
66	message header display panel
67	message contents display panel
68	tool button
69	status bar
72	base layer
74	second layer
76	third layer
100	ShortcutArray
100A	ShortcutEntry
101	method
102	read shortcuts
103	execute automatic organization rules
104	execute filter rules
105	execute correspondent and bulk mail rules
106	process FolderExcludeList
107	process Shortcut Entries
108	search ShortcutArray
108A	create new ShortcutEntry
108B	set ShortcutAction
108C	build ShortcutEntry
110	process shortcut (method)
111	read folder
112	add shortcut
113	update shortcut
114	delete shortcut
115	set shortcut timer
130	handle timed shortcuts (method)
132	initialize shortcut timer

-continued

List of Reference Numerals	
133	read earliest TriggerDateTime
134	set shortcut timer
135	AddChangeShortcut event
137	shortcut timer event
138	perform TriggerAction
140	read folder contents (method)
142	determine if request is initial
142A	check sort direction
143	set position
143A	position at first shortcut in folder
143B	position at last shortcut in folder
144	read shortcuts
145	determine if end of folder reached
146	determine if shortcut excluded from view
147	determine whether enough shortcuts read
148	add shortcut to data structure
149	sort
150	update SortKey (method)
152	read folder data
154	position to read shortcut
155	read shortcut
156	check for end of folder
157	determine sort column data type
158, 158A	determine if new SortKey needed
159	write new SortKey
160	update folder data
217	AddressArray
218	StateCounters
219	StateFlags
220	build memory structures (method)
221	detect if message unsent
222	classify as correspondence
223	initialize data structure
224	retrieve AddressList elements
225	determine whether all elements retrieved
226	initialize AddressEntry
227	increment state counters
228	read address from catalog database 28
229	determine if address found
230	build AddressEntry
231	set state flags
235	process AddressArray (method)
236	get next element of AddressArray
237	determine if end of array reached
238	apply rules
239	determine if shortcut to be created
240	determine if address exists in catalog database 28
240A	add address to catalog database 28
241	determine if status of correspondent should be upgraded

-continued

List of Reference Numerals	
241A	upgrade correspondent status
242	add shortcut
243	determine if shortcuts created
243A	add to unsorted folder
243B	flag as correspondent message
260	move address (method)
261	determine if address to be moved to Unsorted folder
262	delete address
263	read address
264	determine if read successful
265	create address record
266	update address record
267	build AddressQueue
270	process AddressQueue (method)
271A	position at first message summary
271B	read next message summary
271C	determine if all message summaries read
272	determine if message affected by address changes
273	process shortcuts
274	empty AddressQueue

DETAILED DESCRIPTION

[0058] This invention relates to the organization and management of electronic messages. **FIG. 1A** shows a very simple computer network **14** which connects a server computer **12** and two user computers **16** and **18**. The simple network of **FIG. 1A** is an example of one context in which the invention could be practised. Server computer **12** runs messaging server software. In this example, the messaging server is an e-mail server. However, this invention is not limited to e-mail messaging. Those skilled in the art will appreciate that the invention could be applied equally well to other types of messages or to messaging in a mixed environment handling different types of messages.

[0059] Each of user computers **16** and **18** run messaging client software. Computers **16** and **18** can exchange electronic mail messages by way of server **12** and network **14**. For example a user of computer **16** may use e-mail client software to compose a message addressed to a user of computer **18**. When the message is complete the user indicates to the e-mail client software that the message should be sent, for example by activating a "send" icon. Computer **16** then sends the message to server **12** on network **14**. Server **12** receives the message, parses the address and forwards the message to computer **18**. The message is received at user computer **18** by e-mail client software which places the message in an "Inbox" folder. The user of computer **18** can then read the message, respond to the message, delete the message, move the message to another folder, and so on. Over time the user of computer **18** may receive a large number of electronic messages from the user of computer **16** and others. By way of example, this invention could be applied to help the user of computer **18** to organize, locate and manage such messages.

[0060] In general, this invention provides a system and methods which may be practised on a computer for cataloging, retrieving and/or manipulating electronic messages. The invention can be applied to organizing any sort of electronic messages which are to be temporarily or permanently stored. The electronic messages may comprise, for example one or more types of messages selected from the group consisting of e-mail messages, voice mail messages, digitized facsimile messages, and instant messages. The invention could also be applied to any other present or future types of electronic messages. Messages may have attachments. The attachment may be any kind of attached object including things such as images, sound media, video, executable files, word processing files, web pages, scripts, or the like.

[0061] In preferred embodiments of the invention, a system according to the invention handles incoming messages as they are received. However, the invention could also be applied to the organization of messages which already exist in a message store such as an archive containing previously-received e-mail messages. The invention does not rely on any specific message format (such as RFC822 or MAPI) or any specific messaging protocol (such as SMTP or X.400), but can be readily adapted to the set of information made available by any practical message format and protocol.

[0062] **FIG. 1B** shows a schematic overview of the invention. A data store **11** accessible to a processor **13** contains a plurality of messages A, B, C. Processor **13** executes instructions which cause it to create one or more shortcuts **19** to each of the messages in the data store **11**. Data store **11** may comprise a single physical device, may comprise a distributed data store spread over two or more physical devices. Data store **11** may be a single logical device or may comprise multiple logical devices. Each shortcut **19** is associated with a folder. In **FIG. 1B**, horizontally aligned shortcuts **19** are in the same folder. A user interface **15** equipped with a suitable input device **17** permits a user to select a folder and to view and manipulate messages which have shortcuts in the selected folder. The shortcuts have properties which permit large numbers of messages to be effectively handled so that users do not experience unacceptable delays in sorting contents of folders in various ways or while waiting for the contents of a folder to be displayed in interface **15**.

[0063] **FIG. 2** shows in more detail a preferred embodiment of a system **20** according to the invention. System **20** comprises several software components which operate in a computer system. The computer system may comprise one or more intercommunicating computers. For example, the computer system may be user computer **18** of **FIG. 1A**. A collection of electronic messages **22** is stored in one or more message stores **23**. Each message store **23** comprises a memory, file or database structure that provides temporary or permanent storage for the contained messages **22**. A message store server **24** manages the messages **22** in message store **23**. Among other tasks, message store server **24** receives requests for messages **22** from other parts of system **20** and locates and provides the requested messages **22**.

Message store server **24** also notifies other software components of changes to message store **23** by generating events.

[**0064**] Message transport server **26** is a software component that sends and receives electronic messages over a communications network **14** using one or more communications and messaging protocols. Message transport server **26** responds to requests from other software components, and generates events when message transport operations either complete successfully or fail to complete.

[**0065**] System **20** includes a message client **27**. Message client **27** provides a user interface, and receives user input from the interface. The user interface is made available by a suitable input device which typically includes a display and an input device, such as a mouse, trackball, touch screen, keyboard, voice recognition system, or the like. Message client **27** communicates user actions to server software components **24** and **26** by generating and forwarding suitable requests. In turn, message client **27** receives events from server software components that indicate changes that may need to be reflected in the user interface.

[**0066**] All of the foregoing components of system **20** are well known to persons skilled in the art of designing and implementing electronic message-handling systems and, for this reason, do not require further elaboration here. For example, the book *Inside MAPI* by Irving De la Cruz and Les Thaler (1999, Microsoft Press) contains detailed explanations as to how to implement message stores, message store servers (referred to as "message store providers"), message transport servers (referred to as "message transport providers"), message clients (referred to as "MAPI client applications"), and how the related requests and events can be used to coordinate the actions of these software components.

[**0067**] The present invention differs from the other systems described above in various respects. One area of difference is that this invention uses a catalog database **28** and preferably a catalog server **29**, both as described below, to organize the contents of one or more message stores **23**. Catalog database **28** is a data store that contains the information required to organize messages **22** in the associated message stores **23** into a plurality of different folders. Catalog server **29** is a software component that:

[**0068**] implements the organizational algorithms of the invention;

[**0069**] makes outgoing requests to and responds to incoming events from message store server **24**;

[**0070**] responds to incoming requests from and generates outgoing events to message client **27**. Catalog server **29** generates a reply corresponding to each incoming request it receives from another software component (a "requester").

[**0071**] As illustrated in **FIG. 3**, the invention may be applied to automatically associate an electronic message with each of a large number of folders. The association may be made on the basis of any one or more criteria such as date, message status, correspondents, attachments and keywords. An advantage of the invention is that such associations may be made simultaneously on the basis of a wide range of criteria.

[**0072**] As shown in **FIG. 3**, an electronic message **22** typically contains an "envelope"**32** which contains addressing information, a transport header **33** which records how the message was transported, status and organizational information **34** that may be modified by the user, and message contents **36** which may include a message subject, body and zero or more attachments **36**A. The contents of each of these elements may be made the basis for associating a particular message with one or more folders **38**. Due to the complications of working with multiple folders and managing multiple copies of an electronic message it has generally been the case that only a few folders are provided. In preferred embodiments of this invention, however, a very large number of folders may be provided and automatically managed.

[**0073**] The components of the invention may be arranged in various ways on one or more computers without departing from the broad scope of the invention. **FIGS. 4A, 4B** and **4**C show three different example configurations. In a system **40**A having the first configuration, catalog server **29** and catalog database **28** are implemented on the same computer **18** as message client **27**, message store **23** and message store server **24**.

[**0074**] In a system **40**B having the second configuration, message client **27** is implemented on a client computer **18**, while catalog server **29**, catalog database **28**, message store server **24** and message store **23** are implemented on a server computer **12**. Client computer **18** and server computer **12** are connected via communications network **14** which transports the necessary communications (which may be in the form of requests and events) between the message client **27** and the software components running on server computer **12**.

[**0075**] In a system **40**C having the third configuration, message client **27**, catalog server **29** and catalog database **28**, are all located on a client computer **18** while message store server **24** and message store **23** are implemented on server computer **12**. Client computer **18** and server computer **12** are connected via communications network **14** which transports the necessary communications (which may be in the form of requests and events) between the software components running on server computer **12** and the software components running on client computer **18**.

[**0076**] It will be readily apparent to persons skilled in the art that other configurations are possible, including hybrids of the above configurations. It is possible, for example, to provide systems wherein catalog server **29** and catalog database **28** are used to organize the contents of message stores **23** on several separate computers, all connected by a communications network. For clarity message transport server **26** has not been depicted in **FIGS. 4A, 4B** or **4**C. A suitable message transport server would be provided to send and/or receive messages. However, the physical location of the message transport server is not relevant to the invention. The invention may be used in cases where there is no message transport server and no message transport server is required. For example, the invention may be applied to organizing archived electronic messages.

[**0077**] As shown in **FIGS. 5A and 5B** catalog database **28** and message store **23** may be separate from one another or may be integrated in a single integrated message store. Each of these components is preferably provided in the form of a database comprising a plurality of related tables. **FIG. 5A**

shows how a catalog database **28** may be linked to a separate external message store **23**. For the purpose of simplicity, only selected fields in these databases are shown. External message store **23** is identified by a StoreId **51**A, which is a set of information required by message store server **24** to locate and provide access to the contents of the external message store **23**. For example, in a MAPI environment, StoreId **51**A may comprise a MAPI Profile, a valid password, and a MAPI Message Store Entry Identifier. Each message **22** represented in external message store **23** is identified by a StoreMessageId **52**A—which uniquely identifies a message within external message store **23**. Each message **22** may have one or more attachments **36**A associated with it. Each of these attachments is identified by a StoreAttachId—which uniquely identifies an attachment within external message store **23**. StoreMessageId and StoreAttachId may comprise numbers, or other identifiers, assigned to the messages and attachments respectively by message store server **24**.

[**0078**] In **FIG. 5A**, catalog database **28** is linked to external message store **23** as follows. Catalog database **28** has a StoreLink table **51**. Each row in StoreLink table **56** contains the StoreId **51**A of a linked message store **23**. The catalog server **29** can use StoreId **51**A to create a session with the message store server **24** for the linked message store **23**. Catalog database **28** also has a MessageSummary table **52** which contains the StoreMessageId **52**A of messages in message store **23**. MessageSummary table **52** is related to StoreLink table **51** by means of a StoreLinkId foreign key **52**B. In the illustrated embodiment, messages **22** are stored in a Message table **54** in message store **23** and attachments are stored in an Attachment table **55** in message store **23**.

[**0079**] Using the StoreMessageId **52**A and the related StoreId **51**A, catalog server **29** can make requests to the message store server **24** to read messages from message store **23**. Catalog database **28** also has an AttachSummary table **53** which contains a StoreAttachId foreign key **53**A which identifies attachments to messages in message store **23**. Catalog server **29** can use StoreAttachId foreign key **53**A to make requests to message store server **24** for attachments stored in message store **23**.

[**0080**] Catalog server **29** can also use the StoreId **51**A, StoreMessageId **52**A and StoreAttachId **53**A values to map events generated by message store server **24** to the matching row within catalog database **28**. For example, when a message is added to a message store **23**, the message store server **24** assigns a unique StoreMessageId to the message and generates an event which informs catalog server **29** of the newly added message.

[**0081**] It will be readily apparent to a person skilled in the art that the message store and catalog database could be separate from one another, as illustrated in **FIG. 2**, could be united in a single database, or distributed between a number of linked databases. The general scheme described above can be applied to any particular message store **23** and message store server **24**. For example, **FIG. 5B** is an entity-relationship diagram for a storage object **50** which integrates both a catalog database and a message store which is linked to the catalog. The message store is part of the same storage object **50** as the catalog. For the purpose of simplicity, only selected fields in storage object **50** are shown. Storage object **50** has a MessageSummary table **52'** and a Message table **54** which share the same primary key. Similarly, storage object **50** has an AttachSummary table **53'0** and an Attachment table **55'** which share the same primary key. As a result, the mapping of MessageSummary to Message and the mapping of AttachSummary to Attachment is a trivial exercise within a relational database environment.

[**0082**] **FIG. 6** illustrates a possible user interface for use with the invention. The user interface comprises a display **60** which could be used to allow a user to access messages at a desktop or laptop computer. Display **60** has a folder panel **61** in which is displayed a representation **61**A of one or more folders. In the illustrated embodiment, representations **61**A comprise a display of the name of each folder. In the preferred embodiment, representations **61**A indicate the number of unread messages within each folder. In the illustrated embodiment the representation **61**A includes a number in parentheses to the right of the name of the folder, with the number equal to the number of unread messages in the folder. For special purpose folders, such as "Drafts" or "To Do" the number preferably is equal to the total number of messages in the folder instead of the number of unread messages in the folder.

[**0083**] A tabbed dialog **62** permits a user to select which group of folders is displayed in panel **61** by selecting one of the tabs of dialog **62**. A tab of dialog **62** may, for example, identify a subtree of folders defined in catalog database **28**. For example, when the interface detects that a user has selected the date tab **62**A (e.g. by clicking on date tab **62**A, the interface displays a "date" subtree in folder panel **61**. The date subtree contains folders in which messages are arranged by date. For example, the date subtree might contain folders named "Today", "Yesterday", "This Week", "Last Week" each containing appropriately selected messages. When the interface detects that a user has selected the "correspondents" tab **62**B then the interface displays folders in panel **61** in which messages are sorted by correspondent, and so on.

[**0084**] Preferably the interface permits a user to designate and select "Hot" folders. A "Hot" folder is a folder that the user has designated as being "hot" or important to them. Any folder in the system can be made "hot". In the illustrated embodiment, dialog **62** includes a "Hot" tab **62**C. When the interface detects that a user has selected hot tab **62**C, then the interface displays in panel **61** all of the folders that the user has designated as being "Hot". Panel **61** shown in **FIG. 6** displays a list of hot folders.

[**0085**] Interface **60** includes a message list panel **64**. When the interface detects that a user has selected a particular folder, such as the Today folder, the interface displays the contents of the selected folder to be displayed in a suitable format in panel **64**. In the illustrated embodiment, panel **64** displays a list of messages with one row displayed for each message. The fields within the row are displayed as separate columns and are controlled using a column header **65** which has one section for each column. When the interface detects that a user has selected a particular section in column header **65**, the interface sorts the list being displayed in panel **64** by the default sort order for the column (usually in ascending sequence). The interface preferably permits a user to request that the messages be displayed in a reverse order. For example, repeatedly selecting a particular section of column header **65** may cause the interface to toggle between ascending and descending sort orders.

[0086] Display **60** includes a message header display panel **66** and a message contents display panel **67**. When the interface detects that a user has selected a specific message, for example by clicking on a row in the list in panel **64** then the interface displays selected information about the associated message in message header panel **66** and displays the body of the associated message in the message contents panel **67**.

[0087] The interface preferably includes a control which permits a user to select between a first mode in which all messages in a folder can be viewed and a second mode in which only messages from recognized correspondents are available for viewing. In the illustrated embodiment, display **60** includes a tool button **68**. Depending upon whether or not the interface detects that a user has activated tool button **68** the interface either permits all messages in a folder to be displayed in panel **64** or displays in panel **64** only messages from recognized correspondents. If all messages are selected, then a status bar **69** displays a count of the messages in the folder. If correspondent messages only are selected then status bar **69** displays both the number of correspondent messages and the total count of messages in the folder. Preferably, in the second mode, the representation **61**A includes the number of unread correspondent messages in the folder instead of the number of unread messages in the folder.

[0088] Many other types and designs of interface may be used with the invention. Creating such interfaces is within the routine skill and knowledge of those skilled in the art of designing messaging systems and will not be described further. The following description is with reference to an embodiment of the invention in which the message store **23** contains e-mail messages and the computer system **18** has a user interface device capable of supporting a display **60** as shown in **FIG. 6**.

[0089] For the purpose of clarity the detailed operation of the invention will be described with reference to three functional layers, each of which builds upon the functionality of the lower layer(s). **FIG. 7** is a block diagram that shows these functional layers. A base services layer **72** implements the base services needed to organize messages into multiple folders and to communicate with other software components. A second layer **74** provides logic for automatically organizing messages into a number of specific folders. A third layer **76** associates messages with individual correspondents and applies rules to distinguish between correspondents with whom the user of the system has been corresponding directly and others.

[0090] As shown in **FIG. 9**, base layer **72** involves requests and events exchanged between catalog server **29** and other software components as well as methods for processing incoming requests and events and for generating outgoing requests and events. The methods of base layer **72** operate directly on elements of catalog database **28**.

[0091] Catalog database **28** may be implemented using any suitable database manager software. For example, any one of a number of commercial industry-standard database managers could be used to implement catalog database **28**. Catalog database **28** comprises a number of tables. In the illustrated embodiment of the invention there is a Folder table **56**, a Shortcut table **57**, a MessageSummary table **52**, and an AttachSummary table **53**. The relationships between these tables are as follows:

[0092] one-to-many between Folder table **56** and Shortcut table **57**,

[0093] one-to-many between MessageSummary table **52** and Shortcut table **57**,

[0094] one-to-many between AttachSummary table **53** and Shortcut table **57**, and,

[0095] one-to-many between MessageSummary table **52** and AttachSummary table **53**.

[0096] In addition to the tables and fields shown in **FIG. 8**, which are more fully documented below, catalog database **28** contains tables and fields of either **FIG. 5A** or **FIG. 5B** depending on whether catalog database **28** is separate from or integrated with message store **23**.

[0097] A description of the tables and fields used in base layer **72** follows. Folder table **56** preferably implements a hierarchical folder-tree structure. A folder is an abstract organizational entity. Each folder **38** can contain one or more shortcuts to messages in message store **23**. Preferably, each folder **38** can contain other folders. The primary key for folder table **56** is FolderId. This key uniquely identifies a row in Folder table **56**. This key may be generated automatically by catalog server **29** when a new folder **38** is created.

[0098] AlternateKey**1** is ParentFolderId. This key contains a value which is the FolderId for a folder which contains the folder in question. The ParentFolderId can be used to implement a hierarchical folder-tree structure using techniques that are well known in the art. Duplicates are allowed.

[0099] AlternateKey**2** is ParentFolderId+Uppercase(FolderName). This key is used to read a Folder by its FolderName, and to detect and prevent duplicate FolderName values at any given level of the folder tree. Duplicates are not allowed.

[0100] AlternateKey **3** is IsHot. This key is used to mark "hot" folders for the purpose of displaying them to the user in a "hot" view. The key is not populated if IsHot is False. Duplicates are allowed.

[0101] The fields of Folder table **56** are:

[0102] FolderId A non-zero value that uniquely identifies a row in the Folder table.

[0103] ParentFolderId The FolderId of the current Folder's parent in the hierarchical folder tree structure. The root Folder has a ParentFolderId of zero.

[0104] FolderName The display name of the Folder.

[0105] FolderType The type of Folder. Values may include, for example, ftRoot—which indicates that the folder is the root folder, ftSubRoot—which indicates that the folder is a sub-root folder which depends directly from the root folder, (it is convenient to place a category of folders within a sub-root folder, for example, all folders which contain messages selected by date and time could be contained within a "Date" sub-root folder as shown in **FIG. 17**), ftStatus—which indicates that the folder includes messages selected according to a criterion relating to their status, ftCorresp—which indicates that the folder includes messages selected according to a criterion relating to their correspondent, ftMe—which indicates that the folder

includes messages originating from a user of the system (who may have and use multiple e-mail accounts to originate those messages), ftBulkMail—which indicates that the folder includes messages which have been identified as originating from a specific bulk mail sender (such as a mailing list), ftunsorted—which indicates that the folder includes messages for which no correspondent or bulk mail relationships have been established, ftKeyword—which indicates that the folder includes messages selected according to a criterion relating to their keywords, ftDate—which indicates that the folder includes messages selected according to a criterion relating to their dates, ftAttach—which indicates that the folder includes messages selected according to a criterion relating to their attachments, ftSearchResult—which indicates that the folder contains search results and ftuser —which indicates that the folder is user defined.

[0106] IsHot Indicates whether the Folder is "hot" and should be displayed in a preferential manner to the user.

[0107] FolderDateTime The date and time the Folder was created.

[0108] FolderSortColumn The column on which the Shortcuts within a Folder should be sorted. Values depend upon which columns are displayed to the user by message client **27** and may include values such as scIsUnread—which causes messages in the folder to be sorted so that all unread messages are displayed together, scDisplayNames—which causes messages to be sorted alphabetically by the display name of the message sender or message recipient, scSubject—which causes messages to be sorted alphabetically by subject, scMessageDateTime—which causes messages to be sorted by their date and time, scAttachName which causes messages to be sorted according to the names of their attachments, scAttachSize.—which causes messages to be sorted according to the size of their attachments.

[0109] FolderSortDirection The direction in which a-column is sorted. Values are sdascending, sdDescending.

[0110] ShortcutCount The number of shortcuts in the folder.

[0111] UnreadCount The number of unread shortcuts in the folder.

[0112] CorrespShortcutCount The number of correspondent shortcuts in the folder.

[0113] UnreadCorrespCount The number of unread correspondent shortcuts in the folder.

[0114] Shortcut table **57** is a database structure which is used to provide lightweight message shortcuts. Each row in shortcut table **57** associates a MessageSummary or an AttachSummary with a folder **38**. A single MessageSummary or AttachSummary may be simultaneously associated with many folders **38**.

[0115] The PrimaryKey for shortcut table **57** is MessageId+AttachId+FolderId. This key uniquely identifies a shortcut. This key also permits the association of multiple shortcuts with a single MessageSummary row by means of the MessageSummary table's PrimaryKey (MessageId). This key also permits the association of multiple shortcuts with a single AttachSummary row by means of the AttachSummary table's PrimaryKey (MessageId+AttachId).

[0116] AlternateKey**1** for shortcut table **57** is FolderId+SortKey. This key permits the association of multiple shortcuts with a folder by means of the folder table's PrimaryKey (FolderId). This key is also used to sort and read the shortcuts in a folder. Duplicates are allowed.

[0117] AlternateKey**2** for shortcut table **57** is TriggerDateTime. This key is used to implement timed shortcuts that cause an action to be executed when the TriggerDateTime is reached. The key is not populated if TriggerDateTime is zero. Duplicates are allowed.

[0118] In this preferred embodiment of the invention, the structure of shortcut table **57** does not permit multiple shortcuts for the same message to be associated with a single folder. Rather, each shortcut associated with a message must appear in a separate folder.

[0119] The fields of shortcut table **57** are as follows:

[0120] MessageId A non-zero value that uniquely identifies a row in the MessageSummary table.

[0121] AttachId A value that uniquely identifies an AttachSummary within a MessageSummary. AttachId is set to zero if the shortcut is not associated with an AttachSummary.

[0122] FolderId A non-zero value that uniquely identifies a row in folder table **56**.

[0123] SortKey A binary-comparable sort key. This field is described in detail below.

[0124] SortColumn The FolderSortColumn (from folder table **56**) used to construct the SortKey. Values are the same as FolderSortColumn.

[0125] SortDirection The FolderSortDirection (from folder table **56**) used to construct the SortKey. Values are the same as FolderSortDirection.

[0126] TriggerAction Identifies the action to be taken when the TriggerDateTime is reached for a shortcut. Values may include, for example, tanone—to signify that no action should be taken, and taDeleteShortcut-to signify that the shortcut should be deleted when the time specified by TriggerDateTime has been reached. The design can easily be extended to support other actions. For example, as will be readily apparent to persons skilled in the art, timed shortcuts are a powerful general purpose mechanism that could be applied in many applications within a messaging environment (e.g., setting one or more reminders for a message).

[0127] TriggerDateTime The date/time that the action defined in TriggerAction should be executed. A value of zero indicates that no timed action is to be executed for the shortcut. This field may, for example, contain an unsigned 32 bit value that is calculated by multiplying the number of days since Jan. 1, 1980 by 131,072 and then adding the number of seconds since midnight. This permits an action to be triggered at a time specified within a one second resolution.

[0128] IsUnread Matches the IsUnread value from the associated MessageSummary.

[0129] IsCorresp Matches the IsCorresp value from the associated MessageSummary.

[0130] IsUserShortcut Indicates whether the shortcut was created by the user. This permits such shortcuts to be distinguished from automatically created shortcuts. In the preferred embodiment of the invention, user created shortcuts are not deleted when processing automatic organization rules for a message.

[0131] MessageSummary table **52** contains a summary of information about messages and may act as a link to the underlying message objects in message store **23**. The PrimaryKey for MessageSummary table **52** is MessageId. This key uniquely identifies a MessageSummary row. A first reason for implementing a MessageSummary table that is distinct from Message table **54** is that having a separate MessageSummary table **52** facilitates linking to messages in an external message store **23**. A second reason is to enhance processing performance—a MessageSummary record (i.e. a row in MessageSummary table **52**) is typically much smaller than its related message record (i.e. a row in Message table **54**) and it is consequently much faster to read MessageSummary records than it is to read Message records. The actual contents of the MessageSummary will vary depending upon the particulars of the implementation, but in a currently preferred embodiment of the invention MessageSummary table **52** includes the following fields:

[0132] MessageId A non-zero value that uniquely identifies a row in MessageSummary table **52**.

[0133] FolderExcludeList (Array of [FolderId+FolderDateTime])—Identifies the Folders from which shortcuts, that would normally be created by automatic rules, should be removed.

[0134] IsUnread Indicates whether the message is in an unread state.

[0135] IsCorresp Indicates whether the message is correspondence from or to a recognized correspondent. The logic to set this value is described below.

[0136] MessageDateTime The date and time to be displayed to the user when listing messages in a folder. For unsent messages this contains the creation date/time, for received messages this contains the receive date/time, and for sent message this contains the send date/time.

[0137] DisplayNames The sender or recipient names to be displayed to the user when listing messages in a folder. For received messages this contains the sender's name, and for sent or unsent messages this contains is the names of the primary recipients.

[0138] Subject The subject of the message.

[0139] AttachSummary table **53** contains a summary of Attachment information, and may act as a link to the underlying attachment objects in message store **23**. The PrimaryKey for the AttachSummary table **53** is MessageId+AttachId. This key uniquely identifies a row in AttachSummary table **53**. A first reason for implementing an AttachSummary table **53** that is distinct from Attachment table **55** is to facilitate establishing links to attachments **36**A in an external message store **23**. A second reason is to enhance processing performance—an AttachSummary record is typically much smaller than its related attachment record and it is consequently much faster to read AttachSummary records than attachment records. The actual contents of the AttachSummary will vary depending upon the particulars of the implementation, but in a currently preferred embodiment, AttachSummary table **53** has the following fields:

[0140] MessageId A non-zero value that uniquely identifies a row in MessageSummary table **52**.

[0141] AttachId A non-zero value that uniquely identifies an AttachSummary within a MessageSummary.

[0142] AttachType The type of the attachment for organizational purposes. For e-mail attachments this could be a file extension (e.g. "DOC", "EXE", "TIF", "HTML", "VSB") for a file-based attachment, or a string such as "Attached Messages" for an attached message.

[0143] AttachName The display name of an attachment. For e-mail attachments this could be the filename of a file-based attachment, or the subject of an attached message.

[0144] AttachSize The size of the attachment (may be expressed in bytes).

[0145] **FIG. 9** shows the incoming events and incoming requests processed by catalog server **29**, outgoing requests made by catalog server **29**, and outgoing events generated by catalog server **29**. For clarity, interactions between message client **27** and other components are not shown. Message client **27** will typically generate requests in response to user input such as requests to message store server **24** to add, change or delete a message.

[0146] Catalog server **29** handles the incoming requests listed in Table I. If any Request cannot be processed as described, the ResultCode is set to indicate the error that occurred. The Requests listed in Table I are sufficient to implement a working catalog server **29**. It will be apparent to persons skilled in the art that additional requests could be implemented to improve ease of use.

TABLE I

INCOMING REQUESTS TO CATALOG SERVER

Incoming Request	Processing	Reply
AddFolder = ParentFolderId + <Folder properties>	Read parent folder from catalog database 28; Build a new folder using <Folder properties>; Add the folder to catalog database 28; Generate FolderAdded event Build reply.	= ResultCode
ChangeFolder = FolderId + <Folder properties>	Read Folder from catalog database 28; Modify folder using <Folder properties>; Update folder in catalog database 28; Generate FolderChanged event;	= ResultCode

TABLE I-continued

INCOMING REQUESTS TO CATALOG SERVER

Incoming Request	Processing	Reply
	If IsHot has been modified, generate a FolderHotChanged event; Build reply.	
DeleteFolder = FolderId	Read folder from catalog database 28; Verify that folder has no child folders; Verify that folder contains no shortcuts; Delete folder from catalog database 28; Generate FolderDeleted event; Build reply.	= ResultCode
ReadFolder = FolderId	Read folder from catalog database 28; Build reply.	= ResultCode + Folder
ReadFolderSubtree = ParentFolderId	Read all folders from catalog database 28 with matching ParentFolderId (using AlternateKey1); Build reply.	= ResultCode + FolderCount + Array of Folder
ReadHotFolders	Read all folders from catalog database 28 with IsHot = True (using AlternateKey3); Build reply.	= ResultCode + FolderCount + Array of Folder
ReadFolderContents = FolderId + SortKey + SortDirection + RequestCount + CorrespOnly	See FIG. 14 and description below; Build reply.	= ResultCode + SortKey + IsEOF + ContentCount+ ContentArray = Array of [MessageSummary + AttachSummary]
ChangeFolder-SortKey = FolderId + FolderSortColumn + FolderSortDirection	See FIG. 15 and description below; Build reply.	= ResultCode
AddShortcut = FolderId + MessageId + AttachId	Read shortcut from catalog database 28. If shortcut not found then: Read MessageSummary from catalog database 28; If AttachId is non-zero read AttachSummary from catalog database 28; Read folder from catalog database 28; Set IsUserShortcut to True; Empty ShortcutArray (FIG. 10A); Do "AddChangeShortcut" (FIG. 10C); Do "AddShortcut" (FIGS. 11A and 11B); Build Reply	= ResultCode
DeleteShortcut = FolderId + MessageId + AttachId	Read shortcut from catalog database 28. If shortcut not found then: Exit. If IsUserShortcut then Do "DecrementFolderCounts" (FIGS. 11A and 11B); Delete shortcut from catalog database 28; Generate ShortcutDeleted event; Do "UpdateFolder" (FIGS. 11A and 11B); If NOT IsUserShortcut then: Read MessageSummary from catalog database 28; Add the FolderId and FolderDateTime to the FolderExcludeList; Update the MessageSummary in catalog database 28; Process shortcuts as shown in FIG. 10B; Build reply.	= ResultCode
DeleteFolder-Shortcuts = FolderId	Read folder from catalog database 28 and for each shortcut in the folder do: Read shortcut from catalog database 28; Do "DecrementFoldercounts" (FIGS. 11A and 11B); Delete shortcut from catalog database 28; and, Generate ShortcutDeleted event; Do "UpdateFolder" (FIGS. 11A and 11B); Build reply.	= ResultCode
ReadMessage-Summary = MessageId	Read MessageSummary from catalog database 28; Build reply.	= ResultCode
ReadAttach-Summary = MessageId + AttachId	Read AttachSummary from catalog database 28; Build reply.	= ResultCode + AttachSummary

[0147] Table II lists incoming events handled by catalog server **29**.

TABLE II

EVENTS HANDLED BY CATALOG SERVER

Incoming Event	Processing
MessageAdded = StoreId + StoreMessageId	Read message from message store server 24 (using a ReadMessage request); Build a new MessageSummary; Add the MessageSummary to catalog database 28; For each attachment in the message, build a new AttachSummary and add the AttachSummary to catalog database 28 Process shortcuts (FIG. 10B).

TABLE II-continued

EVENTS HANDLED BY CATALOG SERVER

Incoming Event	Processing
MessageChanged = StoreId + StoreMessageId	Read message from message store server 24 (using a ReadMessage request); Read the MessageSummary from catalog database 28 using the StoreId and StoreMessageId; Modify the MessageSummary to match the message; Update MessageSummary in catalog database 28; Delete all AttachSummaries for the message from catalog database 28; For each attachment in the message, build a new AttachSummary and add the AttachSummary to catalog database 28; Generate a SummaryChanged event; Process shortcuts (FIG. 10B)
MessageDeleted = StoreId + StoreMessageId	Read message from message store server 24 (using a ReadMessage request); Read the MessageSummary from catalog database 28 using the StoreId and StoreMessageId; Delete the MessageSummary from catalog database 28; Delete all AttachSummaries for the message from catalog database 28; For each shortcut in the message do the following: Read related folder from catalog database 28; Do "DecrementFolderCounts" (FIGS. 11A and 11B); Delete shortcut from catalog database 28; and, Generate ShortcutDeleted Event; Do "UpdateFolder" (FIGS. 11A and 11B).

[0148] The only Outgoing Request that catalog server **29** must provide in the preferred embodiment of the invention is a ReadMessage request directed to message store server **24**. The formatting of the request will vary depending on the message store server implementation, but for the purposes of this design it is assumed that the ReadMessage Request returns all messages and attachments requested by catalog server **29**.

[0149] Catalog server **29** generates the events listed in Table III.

TABLE III

EVENTS GENERATED BY CATALOG SERVER

Outgoing Event	Processing
FolderAdded = Folder	Generated when a new folder is added to catalog database 28 so that message client 27 can add the folder to the user interface.
FolderChanged = Folder	Generated when a folder is updated in catalog database 28 so that message client 27 can update the folder in the user interface.
FolderHotChanged = Folder	Generated when the value of IsHot changes for a folder so that message client 27 can add or remove the folder from the "hot" view.

TABLE III-continued

EVENTS GENERATED BY CATALOG SERVER

Outgoing Event	Processing
FolderDeleted = Folder	Generated when a folder is deleted from catalog database 28 so that message client 27 can remove the folder from the user interface.
ShortcutAdded = Shortcut	Generated when a shortcut is added to catalog database 28 so that message client 27 can add information about the related message or attachment to the user interface.
ShortcutDeleted = Shortcut	Generated when a shortcut is deleted from catalog database 28 so that message client 27 can remove information about the related message or attachment from the user interface.
ShortcutSortKeyChanged = Shortcut	Generated when the SortKey is changed for a shortcut (except when processing a ChangeFolderSortKey request) so that message client 27 can resequence information about the related message or attachment in the user interface.
SummaryChanged = MessageSummary	Generated when a MessageSummary is changed in catalog database 28 so message client 27 can update information about the related message or attachment in the user interface.

[0150] Those skilled in the art will understand from the foregoing description that shortcut table **57** plays a key role in associating messages with folders. The term "lightweight message shortcut" may be used to refer to the information in a row in shortcut table **57**. A lightweight message shortcut may reside in the shortcut table and may also reside in a data structure in the catalog server **29** or the message client **27**. A lightweight message shortcut may also be present in requests or events communicated between catalog server **29** and other software components.

[0151] The provision and use of lightweight message shortcuts (which are called simply shortcuts in this specification) is a key aspect of this invention. The following design criteria should be satisfied in order to successfully use lightweight message shortcuts. Significant deviation from these criteria can result in unacceptable performance or functionality:

[0152] lightweight message shortcuts must be very small in size, preferably less than 64 Bytes and most preferably less than 32 Bytes (in the currently preferred embodiment of the invention, shortcuts are 24 Bytes);

[0153] shortcut table **57** should have no more alternate keys than are necessary (in the preferred embodiment of the invention the shortcut table has two alternate keys);

[0154] shortcut table **57** should support sorting message summaries by any desired SortColumn;

[0155] shortcut table **57** should support incremental reading, so that the entire contents of a folder do not have to be read before they can be displayed to the user; and,

[0156] shortcut table **57** should have full referential integrity with Folder table **56**, MessageSummary table **52** and AttachSummary table **53**, so that an

action taken on a message or an attachment through a shortcut in one folder **38** will be visible in all folders **38** in which shortcuts to the same message exist.

[**0157**] Preferably the total number of shortcuts in a folder and the number of unread shortcuts in a folder is be available at all times for display in the user interface. This information can be displayed so that a user can look at a list of folders and see how many unread messages are in each folder.

[**0158**] **FIGS. 10A through 10D**, **11**A and **11**B illustrate a process by which lightweight message shortcuts corresponding to a message can be added, updated or deleted. The process makes use of a memory-based ShortcutArray **100** which contains ShortcutEntries **100**A. Each ShortcutEntry **100**A contains the following:

[**0159**] ShortcutAction—possible values are (saAdd—indicating that the shortcut should be added to catalog database **28**, saDelete—indicating that the shortcut should be deleted from catalog database **28**, saKeep—indicating that the shortcut should be retained unaltered in catalog database **28**);

[**0160**] Excluded—which indicates whether or not the shortcut is a user-defined shortcut that should be excluded from being deleted by automatically executing shortcut handling rules;

[**0161**] OldShortcut;

[**0162**] Shortcut; and,

[**0163**] AttachSummary (needs to be present only if the shortcut is for an attachment and ShortcutAction is saAdd or saKeep).

[**0164**] **FIG. 10B** shows a method **101**, which may be performed periodically, for automatically creating a group of shortcuts associated with a message : Method **101** begins by reading all existing shortcuts for a message into a ShortcutArray **100** (as OldShortcut objects) and setting the ShortcutAction to saDelete for each ShortcutEntry **100**A (step **102**). Method **101** continues (step **103**) by applying organizational rules (as described below) to generate a set of new shortcuts for the message. The new shortcuts are saved as Shortcut objects in ShortcutEntrys **100**A in ShortcutArray **100**. Base layer **72** provides an "AddChangeShortcut" method (see **FIG. 10C**). This method is used to process each shortcut in the set of new shortcuts.

[**0165**] The "AddChangeShortcut" method searches the Shortcut Array to determine whether the shortcut being created or changed already exists (step **108**). The method either creates a new ShortcutEntry and sets ShortcutAction to saAdd(step **108**A) or updates an existing ShortcutEntry and sets ShortcutAction to saKeep (step **108**B) depending on whether the shortcut is new with respect to the prior contents of the ShortcutArray or already exists in the ShortcutArray. Finally a ShortcutEntry is built and added to the ShortcutArray (step **108**C).

[**0166**] Preferred embodiments of the invention apply filter rules (step **104**) and correspondent and bulk mail rules (step **105**) as described below.

[**0167**] Preferred embodiments of the invention also include a FolderExcludeList which contains a list of folders in which shortcuts should not be automatically added. Method **101** processes the FolderExcludeList (step **106**). Processing the FolderExcludeList preferably includes validating each element in the FolderExcludeList to make sure that the folder to which it refers has not been deleted and another folder with the same FolderId has been subsequently created. This validation may be done, for example, as shown in **FIG. 10D** by reading the folder from catalog database **28** and comparing the FolderDateTime in the FolderExcludeList element with the FolderDateTime for the folder. If a folder listed in an element of the FolderExcludeList is not found in catalog database **28** or the FolderDateTime values are different, then the element contains an invalid reference and is deleted from the FolderExcludeList. For each valid element, the Excluded flag is set to True in the matching ShortcutEntry (if a matching ShortcutEntry exists).

[**0168**] Method **101** concludes by processing each ShortcutEntry in the ShortcutArray and adding, updating or deleting the corresponding shortcut in catalog database **28** depending on the values of ShortcutAction, IsUserShortcut and Excluded. A method **110** which may be used for processing ShortcutEntries is shown in **FIGS. 11A and 11B** (step **107**). During step **107** the folder counts are incremented or decremented as appropriate. For each Shortcut added, changed, or deleted, the ShortcutCount—which indicates a total number of shortcuts in a folder, UnreadCount—which indicates a total number of shortcuts which have the status unread, CorrespShortcutCount—which indicates a total number of shortcuts which relate to messages to or from a recognized correspondent, and CorrespUnreadCount—which indicates a total number of shortcuts which relate to messages to or from a recognized correspondent and which have a status of unread, are correctly updated for the folder with which the shortcut is associated. ShortcutAdded, ShortcutDeleted, ShortcutSortKeyChanged and FolderChanged events are generated as required.

[**0169**] The net result of method **101** is that, with the exception of shortcuts excluded from consideration because they are listed on the FolderExcludeList, only shortcuts created by the organizational rules and shortcuts created by the user exist in catalog database **28** for the message being processed.

[**0170**] Method **110** reads the folder corresponding to the ShortcutEntry from catalog **28** and, depending upon the value of ShortcutAction, whether or not the shortcut is excluded and whether or not the shortcut is a user shortcut, method **110** either adds a shortcut corresponding to the ShortcutEntry (step **112**), updates a shortcut corresponding to the ShortcutEntry (step **113**) or deletes a shortcut corresponding to the ShortcutEntry (step **114**). If a shortcut is added or updated then method **110** also sets the shortcut timer to the earliest TriggerDateTime for any shortcut in catalog **28** (step **115**).

[**0171**] **FIG. 12** illustrates the structure of shortcuts in a currently preferred embodiment of the invention. The size of the Shortcut is minimized by using a 6 byte SortKey and by encoding TriggerDateTime as a 32 bit unsigned integer. The number of alternate keys is minimized. In the preferred embodiment this is done by:

[**0172**] Ordering the fields in the PrimaryKey so that it can be used both as a primary key, as well as a foreign key to associate shortcuts with a message or an attachment;

[0173] Using AlternateKey1 to incrementally read shortcuts within a folder, as well as a ForeignKey to associate shortcuts with a folder; and,

[0174] Populating AlternateKey2 only if TriggerDateTime is non-zero.

[0175] A feature of preferred embodiments of the invention is that the shortcuts include a sort key (SortKey) which is variable and is dynamically rewritten to accommodate sorting on a particular criterion. In the preferred embodiment each folder has FolderSortColumn and FolderSortDirection properties which specify a criterion for use in sorting. The SortKey is written to match the SortColumn and SortDirection of the folder. A sort key is constructed by a software component which takes as input parameters a folder object, a MessageSummary object, and an AttachSummary object (the AttachSummary object is needed only if creating a Attachment Shortcut, otherwise it can be NULL). The SortKey is constructed by determining which MessageSummary or AttachSummary field to use for sorting (based on a value in the FolderSortColumn). The software component identifies a format for the field and selects a corresponding format for the SortKey. If the field type is Enumeration or Boolean, then the SortKey format is also dependent upon whether FolderSortDirection has a value of sdascending or sdDescending.

[0176] The SortKey is initialized, for example by setting all of its characters to binary zeros. The shortcut SortColumn is set to FolderSortColumn and the Shortcut SortDirection is set to FolderSortDirection. Then the format of the SortKey is set as follows:

[0177] For a field which contains a string of characters in Unicode format, UCW1, UCW2 and UCW3 are set to the Unicode character weights of the first three characters in the string;

[0178] For a field which contains a string of characters in ANSI format, C1, C2, C3, C4, C5, C6 are set to the values of the first 6 characters in the string. Alternatively, the ANSI string could be converted to a Unicode string and the SortKey generated as above.

[0179] For a DateTime field, YY is set to the year minus 1950, MM to the month number, DD to the day of the month, hh to the hour based on a 24 hour clock, mm to the minute, and ss to the second.

[0180] For an Enumeration or Boolean field where SortDirection is sdDescending, Value is set to the numeric equivalent of the Enumeration or Boolean value, and YY, MM, DD and hh are set as described for a Date/Time field.

[0181] For an Enumeration or Boolean field where SortDirection is sdAscending, Value is set to the numeric equivalent of the Enumeration or Boolean value, YY is set to 255 minus the year plus 1950, MM is set to 255 minus the month number, DD is set to 255 minus the day of the month, and hh is set to 255 minus the hour based on a 24 hour clock.

[0182] For a field which contains a 32-bit integer value (an Integer32 value), the first character is set to bits 31-24 of the field, the second character is set to bits 23-16 of the field, the third character is set to bits 15-8 of the field, and the fourth character is set to bits 7-0 of the field, where 0 is the least-significant bit and 31 is the most-significant bit. A SortKey for a field containing a 16-bit integer value (an Integer16 value) may be created by converting the Integer16 value to an Integer32 and following the steps above.

[0183] SortKey values for other field types can be constructed in a manner analogous to the examples above.

[0184] The end result of this SortKey encoding is a binary-comparable sort key that provides an approximate ordering of shortcuts within a folder. The ordering is only approximate because it only compares the SortKeys which contain only the most significant parts of the fields under consideration (for example, the first three letters of a unicode string). There will be cases where several shortcuts might have the same value for the SortKey even though the shortcuts have non-identical values in the column being sorted on. Encoding a partial DateTime value in the Enumeration or Boolean SortKey provides for more control breaks for incremental read processing than can be provided by the value of the field itself. In message client software it is also common practice, when sorting messages on a column other than MessageDateTime, to use MessageDateTime as a secondary sort key with most recent messages shown first—this is the rationale for using inverted YY, MM, DD and hh values in the ascending SortKey, and normal YY, MM, DD and hh values in the descending SortKey.

[0185] Preferred embodiments of the invention support timed shortcuts which cause the execution of an action (TriggerAction) when the system clock advances past a time specified in the TriggerDateTime field of the shortcut. The creator of the Shortcut is responsible for supplying the desired TriggerAction and TriggerDateTime. The action may be, for example, deleting the shortcut.

[0186] As shown in FIG. 13, a method 130 for handling timed shortcuts maintains a shortcut timer. The shortcut timer may be a memory location containing a value equal to the earliest TriggerDateTime for the shortcuts under management which is compared periodically (preferably at least once per second) to the system clock of the computer 18 on which the system is operating. In preferred embodiments of the invention the shortcut timer is a software object which can contain a time value and includes software code which compares the time value to the system clock. A shortcut timer event is generated when the shortcut timer contains a value which indicates a time equal to or earlier than the current time as indicated by the system clock.

[0187] When the system is initialized, the shortcut timer is initialized to an inactive state (step 132). Then the first shortcut is read from catalog database 28 using AlternateKey2 of the shortcut table (step 133). This yields the shortcut from catalog database 28 which has the earliest TriggerDateTime value (since AlternateKey2=TriggerDateTime). If there is such a shortcut then the shortcut timer is set to the TriggerDateTime (step 134). If there is no such shortcut then method 130 stops.

[0188] Each time a new shortcut is added to catalog database 28 or a shortcut in catalog database 28 is changed an AddChangeShortcut event 135 occurs. In response to an AddChangeShortcut event 135 step 134 is repeated for the new shortcut. The result is that the shortcut timer always

contains the earliest TriggerDateTime for any shortcut in catalog database **28**. On the occurrence of a shortcut timer event **137** the system reads the shortcut from catalog database **28** and performs the specified TriggerAction (step **138**). The system then executes method **130** starting at step **133** to set the shortcut timer for the next TriggerDateTime.

[**0189**] Preferred embodiments of this invention permit message client **27** to read the contents of a folder incrementally. This enhances performance because it permits the first few messages in the folder to be displayed to a user very quickly without requiring the user to wait until all of the messages in the folder have been read before seeing the first messages. Providing users with very quick access to messages associated with a folder is important for usability of the system.

[**0190**] When a user selects a folder by way of interface device **25**, message client **27** receives the selection and generates a ReadFolderContents request directed to catalog server **29**. As shown in **FIG. 14**, the ReadFolderContents identifies a folder and preferably includes a request count. The request count is a number which is preferably at least as large as a number of rows visible in panel **64**. The initial ReadFolderContents request preferably specifies a NULL SortKey. In response to the ReadFolderContents request catalog server **29** performs method **140** of **FIG. 14**. The end result of method **140** is a reply which includes a sorted array which message client **27** can use to populate the list displayed in panel **64**. The reply also contains a SortKey value which is used by message client **27** to build subsequent ReadFolderContents requests to retrieve the remaining shortcuts in a folder.

[**0191**] As a user uses interface **25** to scroll through the shortcuts to messages which are displayed in panel **64**, message client **27** generates additional ReadFolderContents requests for catalog server **29**. Message client **27** uses the SortKey provided by catalog server **29** with each reply to construct the subsequent ReadFolderContents request. This continues until the user stops scrolling or an IsEOF flag of True (which indicates that the last shortcut has been reached) is returned in a reply.

[**0192**] If the user has requested that the display of messages which are not associated with recognized correspondents be suppressed (for example, by clicking on tool button **68**) then message client **27** sets a value CorrespOnly in the ReadFolderContent request to True.

[**0193**] Those skilled in the art will understand that message client **27** may have a conventional architecture (although it is not so limited) with some minor modifications to facilitate incremental reading. For incremental reading as described herein a message client **27** should be adapted to behave as follows:

[**0194**] for each folder, message client **27** should save the SortKey and the IsEOF flag from the most recent ReadFolderContents reply for that folder. These values can be used to read the next group of shortcuts for that folder.

[**0195**] when message client **27** receives a ShortcutAdded Event for a folder, it should discard the event if the saved IsEOF flag is False and the event SortKey is greater than the saved SortKey (i.e. the event relates to a shortcut that would not yet have been requested by message client **27** for display even if it had been received earlier).

[**0196**] when message client **27** receives a SummarySortKeyChanged event it should reposition the corresponding entry in the list being displayed in panel **64** to match the new sort sequence. If the saved IsEOF flag is False and the event's SortKey is greater than the saved SortKey, then message client **27** should refresh its display by issuing ReadFolderContents requests until it receives a reply having a IsEOF value of True, or the a reply having a SortKey value greater than the saved SortKey.

[**0197**] Method **140** begins by testing to determine whether the ReadFolderContents request is an initial request or is a request to read more shortcuts (step **142**). In the preferred embodiment of the invention this is done by testing to see if the SortKey supplied with the ReadFolderContent request is NULL. If the request is an initial request then the sort direction is checked (step **142**A). After these initial steps the position in the Shortcut table **82** is established using the FolderId, SortKey and SortDirection from the request. If step **142** determines that the request is not an initial request then the SortKey from the request is used to set a position on AlternateKey**1** for shortcut table **57** (step **143**). Otherwise, depending upon the sort direction determined in step **142**A AlternateKey**1** for shortcut table **57** is used to position for retrieval beginning at either the first shortcut (step **143**A) or the last shortcut (step **143**B) in the folder.

[**0198**] Method **140** continues by reading shortcuts (step **144**) in either ascending or descending sequence until the end of folder is reached or a sufficient number of shortcuts has been read. In the illustrated embodiment, this is performed by, after attempting to read the next shortcut in step **144** determining if the end of the folder has been reached (step **145**). If so then the end of folder flag (IsEOF) is set to true and the method concludes. If not then the retrieved shortcut is tested to see if it is excluded from view by any current filter (step **146**). For example, if the shortcut is not associated with a recognized correspondent (in the currently preferred embodiment this is indicated by setting the IsCorrespondent field in the shortcut to contain a value of False) and CorrespOnly is True, then the shortcut is discarded. If this happens then the method repeats step **144** to retrieve the next shortcut. Each time another shortcut which matches any applicable filter criteria has been read then method **140** tests to see whether enough shortcuts have been read (step **147**).

[**0199**] Step **147** causes method **140** to keep reading shortcuts until either—all of the shortcuts in the folder have been read—or—at least the number of shortcuts specified by the request count have been retrieved and the most recently retrieved shortcut has a SortKey different from the SortKey of the previous shortcut. Each time another shortcut is read, if enough shortcuts have not been read, method **140** adds the most recently read shortcut to a reply data structure (step **148**). The reply data structure is preferably located in memory so that it can be rapidly sorted. When enough shortcuts have been retrieved then method **140** sorts the shortcuts in the reply data structure and sends the sorted results to message client **27** in a reply (step **149**). Sorting the reply data structure in memory corrects for the approximate ordering of shortcuts in AlternateKey**1** of shortcut table **57** that can result from truncated values in the SortKey field.

[0200] One skilled in the art will readily understand that method **140** may be optimized to take advantage of efficiencies which may be obtained by reading multiple shortcuts in single database query. When this is done, the FolderId, SortKey and SortDirection are still provided in the request and used to establish a starting position in shortcut table **82**. This permits method **140** to support reading in either ascending or descending sequence. Even if catalog server **29** reads multiple shortcuts from catalog database **28** in one operation, method **140** still does a partial read of the shortcuts in a folder, and generates a reply only when "enough" shortcuts have been read, as described above.

[0201] As described above, the SortKey in shortcut table **57** is used to retrieve shortcuts, in approximate order, from catalog database **28**. To retrieve shortcuts based upon a different sort criterion (e.g. to sort the current folder based upon values in a different column) it is usually necessary to change the value for the SortKey. A typical message client **27** permits a user to select a column and direction in which messages will be sequenced when they are displayed in panel **64**. If the user is simply sorting the current column in reverse order and the column is of a type which uses the same SortKey for sorting in both ascending and descending order (e.g.—of the examples given above the current column does not hold an enumeration or binary value) message client **27** can send a ReadFolderContents Request to catalog server **29** with the appropriate value for FolderSortDirection and with SortKey initialized (e.g. set to NULL). Catalog server **29** will then read the folder contents from the beginning or the end of the folder identified in the ReadFolderContents request as requested.

[0202] However, if the user changes the sort direction of a column which uses different SortKey values for sorting in ascending and descending orders (for example, a column which contains enumeration or binary values) or if the user selects a different column to sort on (for example by clicking on a header **65** or otherwise selecting a column using user interface device **25**) then the SortKey value for each shortcut in the folder will need to be rewritten before the shortcuts in the folder can be read and sorted as described above with reference to **FIG. 14**. As an alternative to writing new SortKey values first and then reading the shortcuts according to a method like that of **FIG. 14**, the system could read all of the shortcuts in the folder into memory and sort them in memory on the basis of the values in the sort column for the folder (thereby avoiding the need to use the SortKey for sorting). Message client **27** may include logic to decide whether or not to defer updating the SortKey when a user signals a desire to sort on the basis of a new column. If message client **27** decides to defer processing the SortKeys, then it will typically generate and send to the catalog server a ChangeFolderSortKey request in response to the user selecting a new folder to view. Deferred processing typically has better performance characteristics when message client **27** and catalog server **29** are on the same computer, but its ability to process a large folder is limited by the amount of available memory in the computer. Immediate processing to change SortKey values typically has better performance characteristics when the message client **27** and catalog server **29** are on separate computers.

[0203] **FIG. 15** illustrates a method **150** performed by catalog server **29** for updating the SortKey values for shortcuts associated with a folder. In the currently preferred embodiment of the invention method **150** is invoked by a ChangeFolderSortKey request from message client **27**. Method **150** iterates through all shortcuts in the folder identified in the ChangeFolderSortKey request and updates the SortKey for each of the shortcuts. While implementing method **150**, catalog server **29** first reads data about the folder from catalog database **28** (step **152**) and positions itself so that the next shortcut for reading is the first shortcut in the folder (for example by using AlternateKey**1**) (step **154**). Then catalog server **29** reads the next shortcut in the folder (step **155**) and tests to see whether the end of the folder has been reached (step **156**). If there is no next shortcut (i.e. the end of the folder has been reached) then method **150** proceeds to step **160** which updates the folder data in catalog database **28**.

[0204] Where the end of the folder has not been reached then catalog server **29** tests to determine the data type of the current FolderSortColumn (step **157**). Depending upon whether or not the data type requires a new SortKey when the sort direction is changed, method **150** proceeds to steps **158** or **158**A in which catalog server **29** tests to determine whether a new SortKey is needed. Steps **158** and **158**A deal with the situation where a SortKey for a shortcut is updated to a value which is greater than the original SortKey. Such shortcuts will be read a second time during the iteration. The shortcut is only updated once, however because the shortcut's SortColumn and SortDirection are now the same as the folder's FolderSortColumn and FolderSortDirection. If step **158** or **158**A determines that a shortcut's SortKey should be updated then method **150** proceeds to step **159** in which new SortKey, SortColumn and SortDirection values are written to the shortcut in catalog database **28**.

[0205] As mentioned earlier, the design and construction techniques for making a message client suitable for use in this invention are well known in the art. This knowledge includes the generation of requests to message store server **24** and catalog server **29** in response to user interactions, and the handling of the replies that are returned for each request. The knowledge also includes the processing of events generated by message store server **24** and catalog server **29** in order to update the user interface to communicate the underlying changes in message store **23** and catalog database **28** to the user.

[0206] The foregoing description provides a basic framework for the construction of systems and methods which permit shortcuts corresponding to messages to be created and associated together in folders. A single message may have associated with it a great many shortcuts. The shortcuts can be associated into folders in such a manner that a user has many different ways to view a set of messages. In preferred embodiments of the invention shortcuts for messages are generated automatically. This can be viewed conceptually as a higher level function provided by second layer **74** (**FIG. 7**).

[0207] Second layer **74** provides services which automatically organize shortcuts to messages into multiple folders based upon the attributes of the messages. Second layer **74** can also be used to provide enhanced filtering of messages and to provide enhanced manual organization of messages.

[0208] Implementing second layer **74** may involve adding certain fields to tables within catalog database **28**. **FIG. 16** shows one possible set of added fields for implementing second layer **74**. A SearchCriteria field in folder table **56**

contains search criteria that are encoded as a stream, and which can be decoded as required.

[0209] Several fields are added to MessageSummary table **52** as follows:

[0210] Direction identifies the direction of the message. Values are (drSend, drReceive).

[0211] SendState indicates the state of an outgoing message.

[0212] Values are (ssUnsent, ssWaitingSend, ssSent).

[0213] IsDeleted indicates whether a message has been logically deleted.

[0214] IsActive indicates whether a message is in an active state. When a message is added to catalog database **28** IsActive is initially set to True.

[0215] IsKept indicates whether message deletion is to be prevented.

[0216] IsTagged indicates whether the message is marked with a tag (or a flag).

[0217] IsToDo indicates whether a follow-up action is required for the message.

[0218] KeywordList is an array of keywords, encoded so that each keyword can be extracted separately.

[0219] When catalog **28** is created it may be populated with an initial set of folders. An example of a possible initial set of folders is the set of folders shown in **FIG. 17** which are not marked with asterisks. Folders in **FIG. 17** which are marked with asterisks are typically created subsequently as they are required. Alternate embodiments of the invention could have different initial sets of and arrangements of folders without departing from the invention. The Folders that are added when catalog database **28** is initialized may be termed "system folders". Table IV describes a set of rules according to which shortcuts are added to the second layer folders including the folders shown in **FIG. 17**. Each system folder preferably has a pre-assigned FolderId so that catalog server **29** can read any desired system folder simply by specifying its FolderId.

TABLE IV

RULES FOR SECOND LAYER FOLDERS

Folder Description	Rules/Notes
A. Status Folders	
Deleted	Create shortcut in "Deleted" folder if IsDeleted is True. Create no other shortcuts for the message.
Active Mail	Create shortcut in "Active Mail" folder if IsActive is True. Note: The "Active Mail" folder is intended as a repository for messages which the user is actively dealing with. The processing for the MessageAdded event sets IsActive to True for each new message. If supported by message client 27, the user can manually set IsActive to False to remove the shortcut from this folder when he/she is finished dealing with an active message.

TABLE IV-continued

RULES FOR SECOND LAYER FOLDERS

Folder Description	Rules/Notes
Drafts	Create shortcut in "Drafts" folder if Direction is drSend and SendState is ssUnsent.
Kept	Create shortcut in "Kept" folder if IsKept is True. Note: The IsKeep flag is designed to prevent the accidental deletion of messages, and requires the appropriate support from message client 27 and message store server 24.
Received	Create shortcut in "Received" folder if Direction is drReceive.
Sent	Create shortcut in "Sent" folder if Direction is drSend and SendState is ssSent.
Tagged	Create shortcut in "Tagged" folder if IsTagged is True. Note: The IsTagged flag is designed for the user to tag messages to which he/she wishes to pay special attention, and requires the appropriate support from message client 27 and message store server 24.
To Do	Create shortcut in "To Do" folder if IsToDo is True. Note: The IsToDo flag is designed for the user to identify messages on which he/she needs to take follow up action, and requires the appropriate support from message client 27 and message store server 24.
Unread	Create shortcut in "Unread" folder if IsUnread is True.
Waiting Send	Create shortcut in "Waiting Send" folder if Direction is drSend and SendState is ssWaitingSend
B. Date Folders	
Today	Create Timed Shortcut in "Today" folder if MessageDateTime is within the current day. The shortcut expires at midnight of the current day.
Yesterday	Create timed shortcut in "Yesterday" Folder if MessageDateTime is within the previous day. The shortcut expires at midnight of the current day.
This Week	Create timed shortcut in "This Week" folder if MessageDateTime is within the current week. The shortcut expires at the end of the current week.
Last Week	Create timed shortcut in "Last Week" folder if MessageDateTime is within the previous week. The shortcut expires at the end of the current week.
<Month Folder>	Convert MessageDateTime to a display date that contains just year and month (e.g. "1999 Nov"). Use the display date to read within the "Date" subtree of the folder table using AlternateKey2. If a folder is not found, then create a folder within the "Date" subtree with a FolderName of the display date. Create a shortcut in the folder.
C. Attachment Folders	
All Attachments	Create a shortcut in the "All Attachments" folder for each AttachSummary in the message. Note that the AttachSummary needs to be saved in the ShortcutEntry for use when building the Shortcut SortKey.

TABLE IV-continued

RULES FOR SECOND LAYER FOLDERS

Folder Description	Rules/Notes
<Attachment Type Folder>	For each AttachSummary in the message, use the AttachType to read within the "Attachment" subtree of the Folder table using AlternateKey2. If a folder is not found, then create a folder within the "Attachment" subtree with a FolderName of the AttachType. Create a shortcut in the folder. Note that the AttachSummary needs to be saved in the ShortcutEntry for use when building the Shortcut SortKey.
D. Keyword Folders	
<Keyword Folder>	For each Keyword in the KeywordList, use the Keyword to read within the "Keyword" subtree of the folder table using AlternateKey2. If a folder is not found, then create a folder within the "Keyword" subtree with a FolderName of the Keyword. Create a shortcut in the Folder.

[0220] Preferred embodiments of this invention automatically organize messages into multiple folders without any user intervention. The organization may be based either on native properties of the message (e.g. MessageDateTime) or on properties over which the user has control (e.g. Keywords which the user assigns to a message), or both. Catalog server may apply automatic organization rules when processing shortcuts for a message as shown in step **103** of **FIG. 10B**. When the rules state that new shortcuts should be created, catalog server **29** uses the "AddChangeShortcut"method (**FIG. 10C**) to create the new shortcuts. An example set of rules for creating shortcuts in the folders of **FIG. 17** is provided in Table IV.

[0221] The invention can also provide message filtering using message filtering rules to determine folders into which a shortcut to a particular message should be, or should not be, put. Various systems for applying filtering rules to messages are known to those skilled in the art. Such systems may be used for applying filtering rules in this invention as well if modified to create shortcuts according to this invention instead of creating multiple separate copies of each message. The filter engine is invoked when processing shortcuts for a message as shown in step **104** of **FIG. 10B**. The filter engine may use the "AddChangeShortcut" method of **FIG. 10C** to create a shortcut in each desired folder.

[0222] Preferably user interface **25** permits a user to manually select a message and add shortcuts to the selected message to a folder of the user's choice. Such shortcuts can be created by generating an AddShortcut request in response to the user's input. Those skilled in the art will be able, in light of the foregoing disclosure, to create a message client which provides a suitable user interface for permitting users to manually create and manage folders, and to add or delete shortcuts from these folders. Message client **27** can generate and use AddFolder, ChangeFolder, DeleteFolder, AddShortcut and DeleteShortcut requests to cause shortcuts to be placed in user defined folders according to a user's input. In the currently preferred embodiment of the invention, user created folders created under a "User Folders" folder. All user created folders are thus placed in the "User Folders"subtree. Advantageously a user can manually organize the same message into multiple folders without making multiple copies of the message.

[0223] Preferred embodiments of the invention include a search engine, which may be, but is not necessarily, incorporated in catalog server **29** for searching catalog database **28** for messages which meet a user's search criteria. The search engine preferably creates shortcuts to messages which satisfy the search criteria and places the resulting shortcuts into a "saved search results" folder. The SearchCriteria field in the Folder table of catalog database provides a place for the search engine to store the search criteria associated with the folder. The search engine can manipulate search results folders through the use of AddFolder, ChangeFolder, DeleteFolder, AddShortcut and DeleteFolderShortcuts requests. In the current embodiment these search results folders are automatically placed in a "Search Results" subtree beneath a system folder named "Search Results". The net result is the ability to create multiple search results folders each of which contains the results of a user initiated search as well as the information (SearchCriteria) needed to re-run the search upon demand (or periodically).

[0224] Preferred embodiments of the invention automatically recognize correspondents and make possible sophisticated organization of messages into folders defined for recognized correspondents. These functions are performed in third logical layer **76**. Third layer **76** functions can also automatically separate "Bulk Mail" from other mail. Third layer **76** operates by associating addresses with folders, and then using this relationship to automatically create shortcuts, addresses and folders.

[0225] The preferred embodiment of the invention automatically creates a folder for each recognized correspondent. All messages received from the correspondent and all messages sent to the correspondent are visible in that correspondent folder. Preferably catalog server **29** does not immediately create a visible folder for every new correspondent that it identifies. In some cases a user will receive a message which has been sent to a large number of others in addition to the user. Catalog server **29** preferably creates a folder for every new correspondent that it identifies but permits a user to keep these folders hidden until an event or pattern of behaviour emerges which identifies the correspondent associated with the folder as being a recognized correspondent. Correspondents who have not yet become recognized correspondents may be termed "pending correspondents". Bulk mail is automatically made visible in the "Unsorted" folder.

[0226] The user is then able to impose their will upon this automatic organization by creating new bulk mail folders, by merging or separating correspondent folders, by merging or separating bulk mail folders, or by changing selected correspondent folders into bulk mail folders and vice versa. The final result is a highly organized set of correspondent and bulk mail folders that involved very little effort on the part of the user to create. New messages continue to be automatically organized into all the appropriate correspondent and bulk mail folders, and new correspondent folders continue to be created automatically as required.

[0227] Third layer **76** uses of the services of second layer **74**, as well as the Direction, SendState and IsDeleted Mes-

sageSummary fields described above. To support third layer **76**, catalog database **28** has some additional fields and tables as shown in **FIG. 18**. An address table **58** contains address information. This table associates an address with a correspondent or a bulk mail folder. A folder may be associated with zero or more addresses. Address table **58** is preferably sparsely populated, and contains entries only for meaningful addresses. It is not necessary to maintain full referential integrity between address table **58** and MessageSummary table **57**. While the invention could be practised with a fully populated address table **58** in which referential integrity is maintained with MessageSummary table **57** by means of a Address-MessageSummary Relationship table it is believed that this would provide significantly degraded performance.

[**0228**] The keys defined for address table **58** in a preferred embodiment of the invention are as follows:

[**0229**] The PrimaryKey is AddressId. This key uniquely identifies a row in Address table **58**.

[**0230**] AlternateKey**1** is Uppercase(AddressString). This key is used to read address table **58** by the AddressString field. Duplicates are not allowed.

[**0231**] AlternateKey**2** is FolderId. This key is used as a foreign key to associate an Address with a Folder. Duplicates are allowed.

[**0232**] The fields of Address table **58** are:

[**0233**] AddressId A non-zero value that uniquely identifies a row within the Address table.

[**0234**] FolderId A non-zero value that uniquely identifies a row within the Folder table.

[**0235**] AddressString The address used by the underlying message transport protocol (e.g., an SMTP address or an X.400 address).

[**0236**] AddressType Identifies the type of address. Values are (atNoAddress—which indicates that there is no matching address in catalog database **28**, atMyAddress—which indicates an address of the user of the messaging system, atCorrespAddress—which indicates an address of a recognized correspondent, atBulkAddress—which indicates an address of a recognized source of bulk mail).

[**0237**] IsPendingCorresp Indicates whether the related folder has been identified as a pending correspondent folder.

[**0238**] The following additional field may be added to Folder table **56** in catalog database **28** to implement the functions of third layer **76**:

[**0239**] IsPendingCorresp Indicates whether the Folder has been identified as a Pending Correspondent Folder.

[**0240**] The following additional fields may be added to

[**0241**] MessageSummary table **52** to implement the functions of third layer **76**:

[**0242**] AddressList (Array of [AddressRole+AddressString+AddressName])—encoded so that each element can be extracted separately.

[**0243**] The components of the AddressList are:

[**0244**] AddressRole Identifies the role of the Address. Values are (arFrom, arSender, arTo, arCc, arBcc). For example, an address in the RFC 822 "From:" header would be assigned an AddressRole of arFrom. Similarly addresses in the "Sender:", "To:", "Cc:" and "Bcc:" headers would be assigned AddressRoles of arSender, arTo, arCc, and arBcc respectively. Similar mappings can be made for other messaging protocols.

[**0245**] AddressString The address used by the underlying message transport protocol (e.g., an SMTP address or an X.400 address).

[**0246**] AddressName The display name associated with an address, if available.

[**0247**] When a catalog database **28** according to the preferred embodiment of the invention is initialized, it contains system folders for "Correspondents", "Me", "Bulk Mail" and "Unsorted" messages as shown in **FIG. 19**. Details of these folders are provided in Folder table **56**. Third layer **76** functions use knowledge of addresses that are associated with the messaging system user to classify messages according to their relationship to the messaging system user. A messaging system user may have multiple addresses. These addresses may be obtained by examining the messaging system's configuration information. In the alternative message client **27** may provide an interface which requests that the user enter his/her address(es) through the user interface. Each of these addresses is added as a row in Address table **58** using an AddMyAddress request.

[**0248**] **FIG. 20** shows incoming requests that may be processed by catalog server **29** for implementing the functions of third layer **76**. The processing for these incoming requests is shown in Table V. If any request cannot be processed as described, the ResultCode is set to indicate the error that occurred.

TABLE V

THIRD LAYER REQUESTS

Incoming Request	Processing	Reply
AddMyAddress = AddressString	Build a new Address using: - FolderId of the "Me" system folder AddressString from request AddressType of atMyAddress Add the address to catalog database 28; Build reply.	= ResultCode
ReadFolder-Addresses = FolderID	Read all Addresses for the Folder using AlternateKey2 Build Reply	= ResultCode + AddressCount + Array of Address
MoveAddress = AddressString + TargetFolderId	Process as shown in FIG. 26 Build Reply	= ResultCode
ProcessAddress Queue	Process as shown in FIG. 27 Build Reply	= ResultCode

[**0249**] Third layer **76** applies a set of rules to automatically create shortcuts to messages and to associate those shortcuts with correspondents. This set of rules is preferably executed by catalog server **29** when processing shortcuts for a message as shown in step **105** of **FIG. 10B**. Catalog server **29** can use the "AddChangeShortcut" method (**FIG. 10C**) to

create new shortcuts. It is advantageous to apply the rules of third layer **76** in two phases. In a first phase information about the addresses in a message is gathered and placed in memory structures. Suitable memory structures are illustrated in **FIGS. 21A, 21B** and **21**C. The memory structures include an AddressArray **217** (**FIG. 21A**), StateCounters **218** (**FIG. 21B**) and StateFlags **219** (**FIG. 21C**). **FIG. 22** shows a possible method **220** for building the memory structures.

[**0250**] The way in which third layer **76** generates shortcuts to a message in bulk mail or correspondence folders depends upon the status of the message. If a message is logically deleted, then second layer **74** (step **103** of **FIG. 10B**) creates a shortcut in the "Deleted" folder and no other shortcuts. Third layer **76** performs no processing for logically deleted messages. If a message is unsent (as detected at step **221**) then third layer **76** creates no shortcuts, and classifies the message as correspondence (as opposed to bulk mail) by setting IsCorresp to True (step **222**).

[**0251**] For all other messages, AddressArray **217** is initialized to an empty state, StateCounters **218** are set to zero, and StateFlags **219** are set to False (step **223**). Then, each element in the MessageSummary's AddressList is retrieved (step **224**), and an AddressEntry is created and added to AddressArray **217** as shown in **FIG. 22**. AddressEntry **217** is processed as follows:

[**0252**] AddressRole, AddressString and AddressName are set from values in the AddressList element (step **226**);

[**0253**] AddressId and FolderId are set to zero and AddressType is set to atNoAddress (step **226**);

[**0254**] IsPendingCorresp, IsConfirmedCorresp and CreateShortcut are set to False (step **226**);

[**0255**] StateCounters **218** are incremented as shown in **FIG. 21B** (step **227**);

[**0256**] The address is read from catalog database **28** (step **228**) using AlternateKey**1**. If the Address is found (step **229**), then AddressId, FolderId, AddressType and IsPendingCorresp are set from values in the Address (step **230**); and,

[**0257**] StateFlags **219** are set as shown in **FIG. 21C** (step **231**).

[**0258**] When all of the elements in the AddressList of the MessageSummary for a message have been retrieved, as determined by step **225**, then AddressArray **217** created by method **220** is processed. this may be done by a method **235** as shown in **FIG. 23**. For each entry in AddressArray **217**, the following processing is performed:

[**0259**] Each entry in AddressArray **217** is retrieved (step **236**);

[**0260**] The correspondent and bulk mail rules shown in **FIGS. 24A and 24B** are applied (step **238**);

[**0261**] If, after applying the rules, the CreateShortcut flag is True, as determined at step **239**, then:

[**0262**] If AddressId is zero, as determined at step **240**, the required address and folder are added to catalog database **28** as shown in **FIG. 25A**"AddAddressFolder" (step **240**A).

[**0263**] Otherwise, if IsPendingCorresp is True and IsConfirmedCorresp is True, as determined at step **241** then the correspondent is upgraded from a pending state, as shown in **FIG. 25B**"UpgradePendingCorresp" (step **241**A).

[**0264**] The requested Shortcut is generated by using the AddChangeShortcut method of **FIG. 10B** (step **242**).

[**0265**] ShortcutCount (**FIG. 21B**) is incremented (step **242**).

[**0266**] When all entries in AddressArray **217** have been processed as determined at step **237**, ShortcutCount is examined to determine if any Shortcuts were generated (step **243**). If no Shortcuts were generated, then a shortcut to the message is added to the "Unsorted" Folder using the "AddChangeShortcut" method of **FIG. 10B**, and IsBulkMail is set to True (step **243**A). Finally, the IsCorresp field in the MessageSummary is set to NOT(IsBulkMail) (step **243**B).

[**0267**] The Correspondent and bulk mail rules applied in step **238** are shown in **FIGS. 24A and 24B**. These rules are applied to each AddressEntry in AddressArray **217**. The functions of the rules can be briefly explained as follows:

[**0268**] Rule B1—This rule creates a shortcut for an existing bulk mail address.

[**0269**] Rule B2—This rule creates a shortcut for an existing correspondent address if a received bulk mail message was also addressed directly to the user by the existing correspondent.

[**0270**] Rule B3—This rule creates a shortcut for an existing correspondent address if a sent bulk mail message was also addressed by the user directly to the existing correspondent.

[**0271**] Rule B4—This rule results in no additional shortcuts being created for a bulk mail message, (other than any shortcuts created by preceding rules).

[**0272**] Rule S1—This rule creates a Shortcut in the "Me" folder for a sent message if the message contains an address belonging to the "Me" folder.

[**0273**] Rule S2—Where a sent message does not contain an address belonging to the "Me" folder, and the message contains a "From" address but no "Sender" address, this rule adds the "From" address to the "Me" folder and creates a shortcut in the "Me" folder. This rule is not required for the successful operation of the invention, but is a convenience if the user changes his/her address. The rule assumes that if the "From" and "Sender" addresses are identical then only the "From" address is present in the message, as is the case with RFC 822 e-mail messages. For messaging systems that populate both the "From" and "Sender" addresses regardless of whether they are identical, it would be necessary to modify the implementation to handle this situation. The preferred solution would be to not add a "Sender" address to the AddressList if the "Sender" address is identical to the "From" address.

[**0274**] Rule S3—This rule creates a Shortcut for the "To" address in a sent message if the message has only a single "To" address and no "Cc" or "Bcc" addresses.

If the address does not exist in catalog database **28** then the address and an associated folder are created as a confirmed correspondent (i.e. IsPendingCorresp is False). If the address and folder exist as a pending correspondent, they are upgraded to a confirmed correspondent.

[**0275**] Rule S4—This rule creates a shortcut for the "To", "Cc" or "Bcc" addresses in a sent message if a shortcut hasn't already been created for the address. If the address does not exist in catalog database **28** then the address and an associated folder are created as a pending correspondent (i.e. IsPendingCorresp is True).

[**0276**] Rule R1—This rule creates a shortcut in the "Me" folder for a received message if the message is addressed to me and contains an address belonging to the "Me" folder.

[**0277**] Rule R2—This rule creates a shortcut for the "From" and "Sender" addresses in a received message if the message is addressed to the user. If the address does not exist in catalog database **28** then the address and an associated folder are created as a confirmed correspondent (i.e. IsPendingCorresp is False). If the address and folder exist as a pending correspondent, they are upgraded to a confirmed correspondent.

[**0278**] Rule X1—This rule creates a shortcut for a existing correspondent where the message is sent by the existing correspondent but is not addressed to the user. The message could either be bulk mail or could be a blind carbon copy to the user—there is no way of determining this for messages where all "Bcc" information is completely suppressed.

[**0279**] IsToMe, IsFromKnownCorresp, IsFromMe, IsToKnownCorresp, are StateFlags **219** as shown in **FIG. 21C**. These flags contain the results of Boolean expressions which respectively indicate: whether the message is addressed to the message system user; whether the message is from a recognized correspondent; whether the message is from the message system user; and whether the message is addressed to a recognized correspondent. The terms OriginCount and ToCount respectively are StateCounters **218** as shown in **FIG. 21B**. These StateCounters contain respectively: the number of senders of a message (a message can have multiple senders if it is sent by one messaging user on behalf of another user) and the number of direct recipients of a message (for e-mail messages this is the same as the number of To: addresses).

[**0280**] Third layer **76** preferably permits a user to manually change the Address/Folder relationships, and then to update catalog database **28** based on these changes. **FIG. 26** shows a method **260** for associating an address with a folder. Method **260** is initiated by a MoveAddress request. The address identified in a MoveAddress request may either already exist in catalog database **28** or be a new address. Method **260** checks to determine if the address is being associated with the "Unsorted" folder (step **261**). A feature of the sparsely populated Address table is that a message with a shortcut in the "Unsorted" folder has no other Correspondent or Bulk Mail shortcuts, and the "Unsorted" Folder has no addresses associated with it. Consequently, MoveAddress processing is preferably performed to delete an address from catalog database **28** (step **262**) if a message is moved to the "Unsorted" Folder.

[**0281**] If the address is being moved so that it will be associated with a folder other than the Unsorted folder an attempt is made to read the address from catalog database **28** (step **263**). If the address is not found, as determined at step **264**, then a record for the address is created in catalog database **28** (step **265**). The address record is then updated in catalog database **28** (step **266**).

[**0282**] The MoveAddress request also adds the AddressString to an AddressQueue (step **267**). The AddressQueue can be processed using the ProcessAddressQueue request as illustrated in **FIG. 27**. The AddressQueue can be a list of AddressStrings that is written to a text file or some other storage medium. Multiple AddressStrings may be added to the AddressQueue so that the processing of the AddressQueue can be deferred if desired.

[**0283**] After a user has associated addresses with particular folders the message system processes the message summaries in catalog database **28** to determine whether the changes to the addresses requires any shortcuts to be added or deleted. This may be done by the method **270** of **FIG. 27**. Method **270** reads all MessageSummaries in catalog database **28**, and for each MessageSummary determines if the AddressList in the MessageSummary contains at least one of the AddressStrings in the AddressQueue (step **272**). If so, shortcuts for the message are processed (step **273**) as shown in **FIG. 10B**, which adds, updates, or deletes Shortcuts based on the current Address/Folder relationships. If the IsCorresp value in the MessageSummary changes as a result of this processing, then the MessageSummary is also updated in catalog database **28** (step **273**).

[**0284**] In method **270** all message summaries are read by positioning at a first message summary (step **271**A) and then reading message summaries sequentially (step **271**B) until it is determined that all message summaries have been read (step **271**C).

[**0285**] Even though the ProcessAddressQueue method reads all MessageSummaries in catalog database **28** its performance can be quite good where, as is strongly preferred, the MessageSummary records are quite small. Small MessageSummary records reduce the time required for physical I/O operations in reading MessageSummary table **52**. Also, no shortcut processing is performed on message summaries that are not affected by the updated Address/Folder relationships (step **272** bypasses such message summaries).

[**0286**] The automatic organization that is performed by third layer **76** is useful and appropriate, but typically requires some additional manipulation by the user. This user manipulation can be performed through industry-standard user interface elements implemented by message client **27**, and then communicated to catalog server **29** using the appropriate requests. An implementation of this invention may permit a user to perform any of many actions such as the following (it is assumed in the following discussion that ProcessAddressQueue Requests are deferred and then sent to catalog server **29** as desired by the user):

[0287] Moving an address from one folder to another can be done with a MoveAddress request to move the address to the target folder;

[0288] Combining two correspondent folders or two bulk mail folders into one can be done by:

[0289] Using the ReadFolderAddresses request to obtain a list of addresses in the first folder; and,

[0290] Moving each of the addresses from the first folder to the second folder using the MoveAddress request.

[0291] Deleting shortcuts from a source folder can be done using the DeleteFolderShortcuts request.

[0292] Deleting a folder can be done using the DeleteFolder request.

[0293] Changing a correspondent folder into a bulk mail folder can be done by:

[0294] creating a new bulk mail folder with the CreateFolder request;

[0295] using the ReadFolderAddresses request to obtain list of addresses in the correspondent folder;

[0296] moving each address to the bulk mail folder using the MoveAddress request;

[0297] deleting the shortcuts from the correspondent folder with the DeleteFolderShortcuts request; and,

[0298] deleting the correspondent folder with the DeleteFolder request.

[0299] Changing a bulk mail folder into a correspondent folder can be done by:

[0300] creating a new correspondent folder with the CreateFolder request;

[0301] using the ReadFolderAddresses request to obtain a list of addresses in the bulk mail folder;

[0302] moving each address to the correspondent folder with the MoveAddress request;

[0303] deleting shortcuts from the bulk mail folder with the DeleteFolderShortcuts request; and,

[0304] deleting the bulk mail folder with the DeleteFolder request.

[0305] Creating a new bulk mail folder can be done with the CreateFolder request.

[0306] Associating an address with a bulk mail folder can be done with the MoveAddress request.

[0307] Removing an address from a bulk mail folder (this would be done, for example, when the message system user no longer wants bulk mail organized based on the address)can be done by moving the address to the "Unsorted" folder with the MoveAddress request.

[0308] The methods of third layer **76** can automatically distinguish a possible (or pending) correspondent from a confirmed correspondent, and automatically upgrade a pending correspondent to a confirmed correspondent when appropriate.

[0309] Preferred implementations of the invention include a computer system programmed to execute a method of the invention. The invention may also be provided in the form of a program product. The program product may comprise any medium which carries a set of computer-readable signals corresponding to instructions which, when run on a computer, cause the computer to execute a method of the invention. The program product may be distributed in any of a wide variety of forms. The program product may comprise, for example, physical media such as floppy diskettes, CD ROMs, DVDs, hard disk drives, flash RAM or the like or transmission-type media such as digital or analog communication links.

[0310] It can be appreciated that the preferred embodiment of the invention described above permits messages to be organized in new ways including :

[0311] automatic organization of messages into multiple folders based on message status, e.g. Active Mail, Deleted, Drafts, Received, Sent, Waiting Send and Unread;

[0312] automatic organization of messages into multiple folders based on the message date, e.g. Today, Yesterday, This Week, Last Week, and Month;

[0313] automatic organization of messages into multiple folders based on attachment type;

[0314] automatic organization of messages into multiple folders based on user assigned values, e.g. Kept, Tagged, To Do, and Keywords;

[0315] support for user overrides of automatic organization by letting the user include a message in or exclude a message from any folder;

[0316] automatic organization of messages into multiple folders based on correspondent;

[0317] automatic separation of bulk mail from correspondence;

[0318] techniques to manage correspondent and bulk mail folders;

[0319] support for multiple saved search results folders; and

[0320] support for improved filtering mechanisms.

[0321] The invention may be applied to organizing e-mail messages. An e-mail client which embodies the invention may provide a unified view of multiple message stores through the use of shortcuts. One useful application is creating a catalog database **28** for both a mailbox containing recent e-mail messages and one or more archived message stores. Use of the preferred embodiment of the invention permits a user to identify potentially important messages among less important messages by separating bulk mail from correspondence and by letting users identify hot folders. Users can manage the flow of messages through use of the ActiveMail folder and the ToDo status folder.

[0322] As will be apparent to those skilled in the art in the light of the foregoing disclosure, many alterations and modifications are possible in the practice of this invention without departing from the spirit or scope thereof. For example, it will be understood from the foregoing that in systems or methods according to various embodiments of the invention:

[0323] the preferred embodiment described above uses an event-driven architecture wherein software components inform other software components of certain of changes by generating events. This is currently considered to be a good way to implement this invention. An event-driven architecture is not fundamental to this invention however. Other software techniques could also be used to coordinate the operation of the various components of this invention.

[0324] The description above has broken the software components used to practise the preferred embodiment of the invention into discrete logical components. In any particular implementation it would be possible to combine multiple logical components into a single component. For example, the functions of message store server **27** and catalog server **29** could be integrated into a single component. Software components which are described herein as being a single component could also be split into multiple sub-components without departing from this invention.

[0325] While much of the functionality of the preferred embodiments described above has been expressed as being provided by the same software that operates as a server for catalog database **28** this is convenient but not necessary to the practice of the invention;

[0326] While the term database has been used in this specification primarily to refer to relational databases this is not required in all cases.

[0327] any number of mechanisms can be implemented in the user interface to let a user manipulate folders and messages, including pop-up menus, accelerator keys, dialog boxes, and drag-and-drop operations.

[0328] While it is not preferred, the same filter engine could be used to apply both automatic organization rules and filtering rules.

[0329] The invention may be applied to many different message types.

[0330] The invention may be used in conjunction with many different user interfaces on many different display devices.

[0331] The names of methods, requests, and events can be changed.

[0332] Accordingly, the scope of the invention is to be construed in accordance with the substance defined by the following claims.

What is claimed is:

1. A method for automatically organizing electronic messages in a computer system, the method comprising:

receiving an electric message having a plurality of properties;

storing the electronic message in a data store;

automatically identifying a plurality of folders with which the electronic message is to be associated by applying rules to the properties of the stored message; and,

automatically generating a plurality of shortcuts to the stored message, each of the shortcuts comprising an element directly or indirectly identifying the stored message and an element directly or indirectly identifying one of the plurality of folders.

2. A method according to claim 1 comprising storing the plurality of shortcuts as records in a database.

3. A method according to claim 2 wherein at least one of the properties comprises a date and the rules include a rule which identifies one folder with which the message is to be associated based upon a comparison of the property's date to a current date.

4. A method according to claim 3 comprising periodically and automatically generating new shortcuts, each of the new shortcuts associating the stored message with a folder based upon a comparison of the property's date to a current date.

5. The method of claim 2 wherein the data store holds a plurality of messages and the method comprises receiving from a user folder selection information identifying a selected one of the plurality of folders, retrieving from the database a plurality of shortcuts associated with the selected folder, each of the shortcuts associated with a different one of the plurality of messages, and displaying a representation of the retrieved shortcuts on a display.

6. The method of claim 5 wherein each of the shortcuts comprises a SortKey value and retrieving from the database a plurality of shortcuts comprises retrieving the plurality of shortcuts in order of the SortKey values of the retrieved shortcuts.

7. The method of claim 6 wherein retrieving each of the plurality of groups from the database comprises sequentially retrieving shortcuts associated with the selected folder from the database in order of the SortKey value until either:

a) a last shortcut associated with the selected folder has been retrieved from the database; or,

b) at least a predetermined number of shortcuts has been retrieved from the database in a current group and the SortKey value for a most recently retrieved shortcut is different from the SortKey value for a second-most-recently retrieved shortcut.

8. The method of claim 1 wherein the folders comprise a plurality of date folders each associated with a different period, the method comprising automatically generating a shortcut to the message in each of the plurality of date folders for which the time associated with the message is between the beginning and the end of the folder's period.

9. The method of claim 8 wherein the date folders comprise a Today folder for which the period is a current day, a This Week folder for which the period is a current

week, a Yesterday folder for which the period is an immediately previous day and a Last Week folder for which the period is an immediately previous week.

10. The method of claim 1 wherein the properties include a time and the method comprises identifying a period corresponding to the time associated with the message and creating a date folder associated with the identified period.

11. The method of claim 1 wherein the record in the database has a length of 64 Bytes or less.

12. The method of claim 1 wherein the record in the database has a length of 32 Bytes or less.

13. The method of claim 1 wherein the record in the database has a length of 24 Bytes or less.

14. The method of claim 1 wherein the message has an attachment and the shortcut additionally comprises an AttachId value identifying the attachment.

15. The method of claim **24** wherein the attachment is stored in a data store and the AttachId value is an address of the attachment in the data store.

16. The method of claim **24** wherein the attachment is stored in a data store and the AttachId value is an address of an AttachmentSummary record in the database, wherein the AttachmentSummary record contains an address for the attachment in the data store.

17. The method of claim **24** comprising identifying a type of the attachment and automatically creating a shortcut to the message in a folder corresponding to the type of attachment.

18. The method of claim **27** comprising automatically creating the folder corresponding to the type of attachment.

19. The method of claim 2 wherein the shortcuts comprise entries in a Shortcut table having a primary key comprising a plurality of ordered fields.

20. The method of claim 19 wherein the primary key comprises MessageId+AttachId+FolderId, where AttachId is a field containing a value which associates the shortcut with an attachment to a message.

21. The method of claim 1 comprising, for a plurality of messages, identifying one or more addresses associated with the message and creating a shortcut to the message in a folder associated with each of the addresses.

22. The method of claim 21 comprising applying the rules of **FIG. 24A** to each of the addresses.

23. The method of claim 1 comprising creating the shortcuts at least in folders associated with each of: a time associated with the message; a sender of the message; and a type of attachment associated with the message.

* * * * *