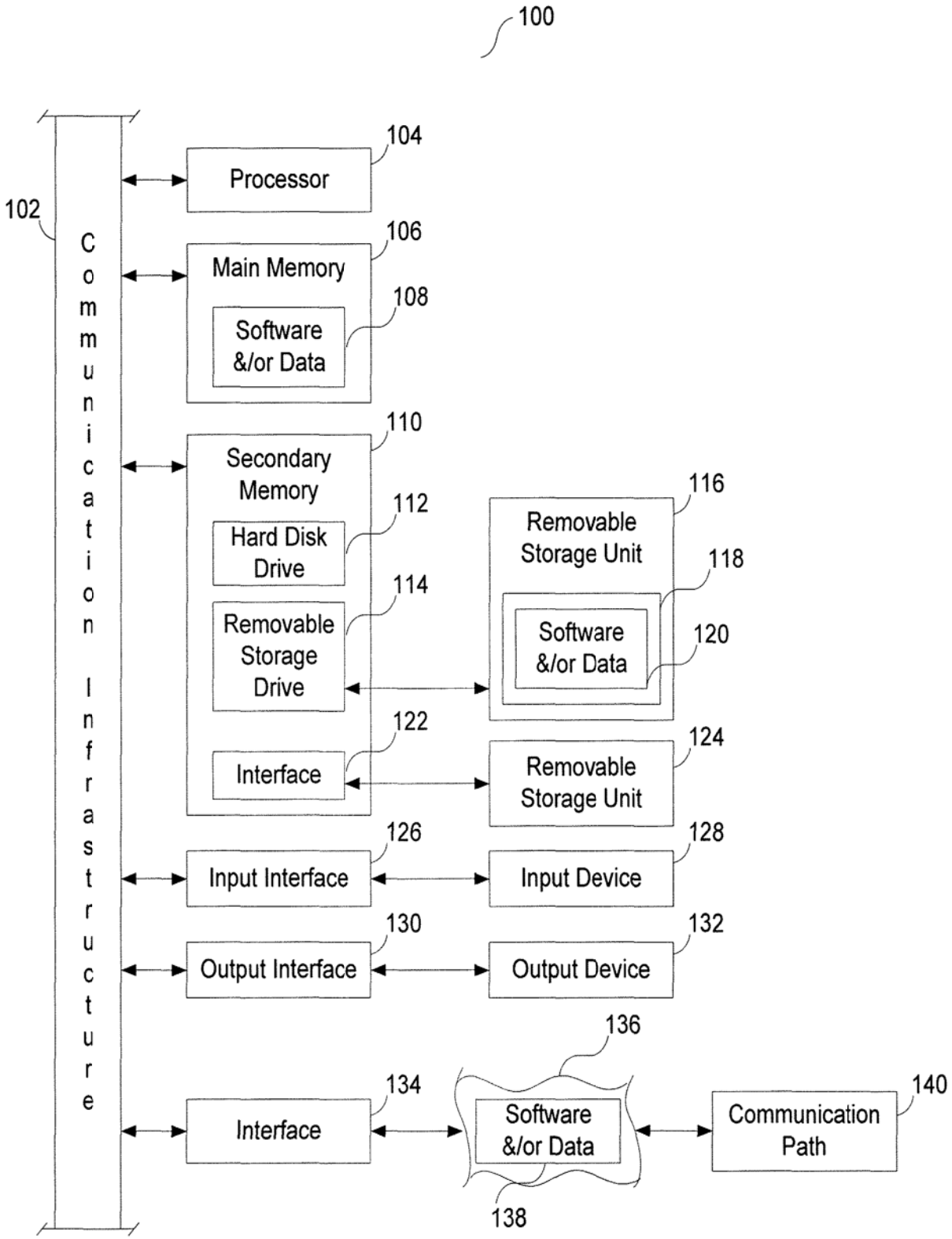
100
102
Communication Infrastructure
Processor
104
Main Memory
106
Software &/or Data
108
Secondary Memory
110
Hard Disk Drive
112
Removable Storage Drive
114
Interface
122
Removable Storage Unit
116
Software &/or Data
118
120
Removable Storage Unit
124
Input Interface
126
Input Device
128
Output Interface
130
Output Device
132
Interface
134
Software &/or Data
136
138
Communication Path
140

FIG. 1

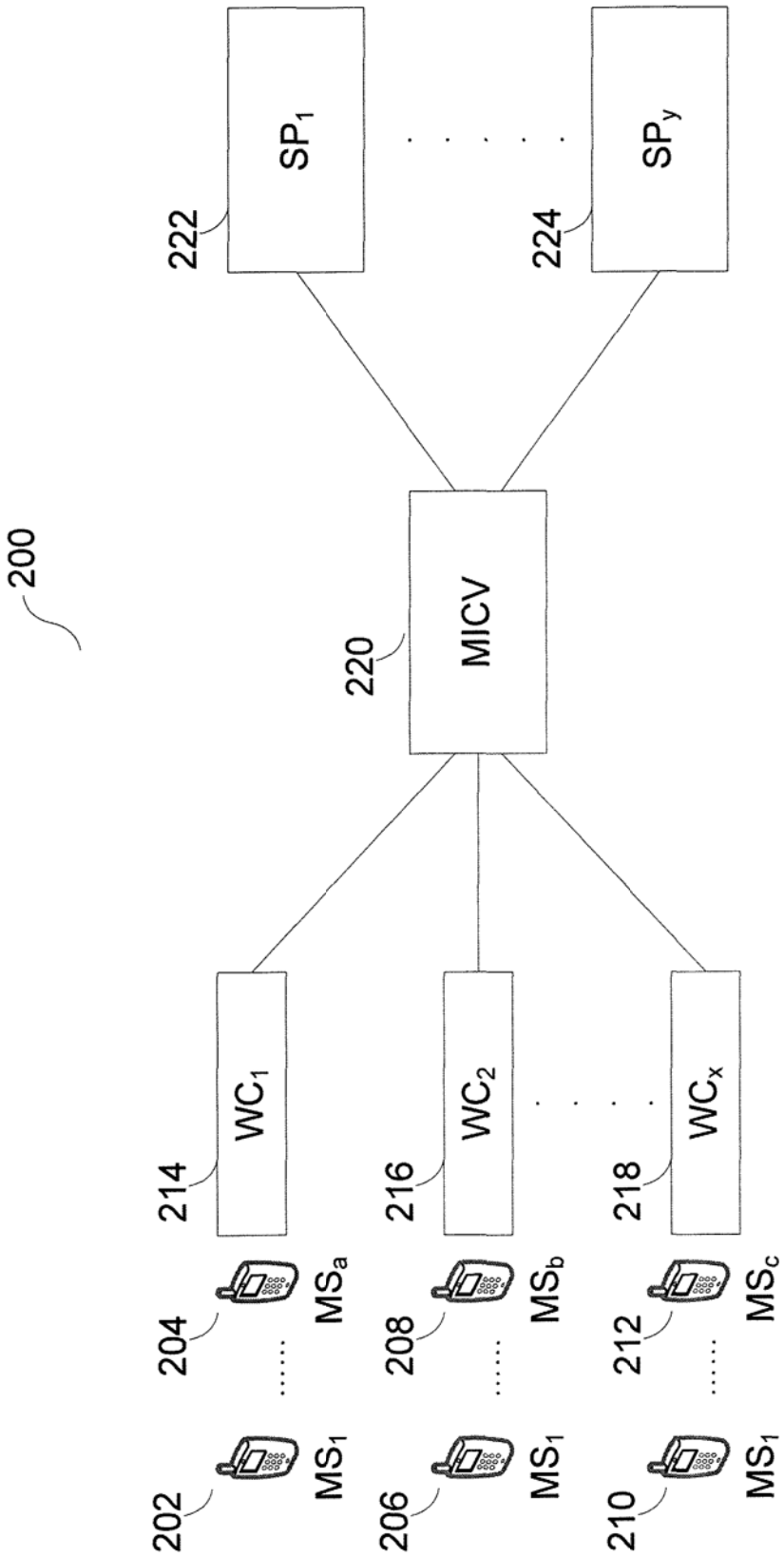
200
220
MICV
222
SP1
224
SPy
214
WC1
216
WC2
218
WCx
202
MS1
204
MSa
206
MS1
208
MSb
210
MS1
212
MSc

FIG. 2

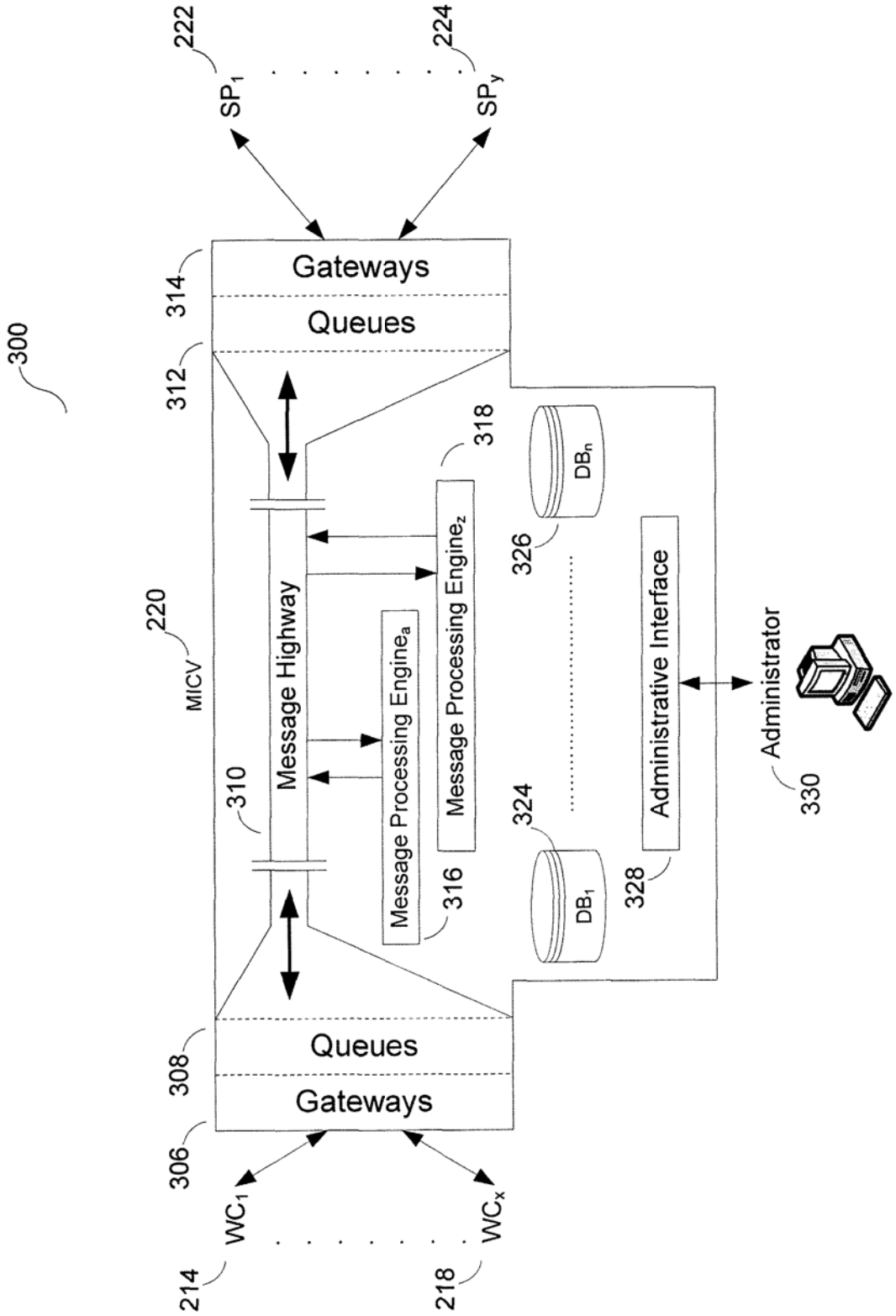
300
222
224
SP1
SPy
Gateways
Queues
314
312
318
DBn
326
220
MICV
Message Highway
Message Processing Engine z
Message Processing Engine a
310
316
324
DB1
328
Administrative Interface
Administrator
330
Queues
Gateways
308
306
WC1
WCx
214
218

FIG. 3

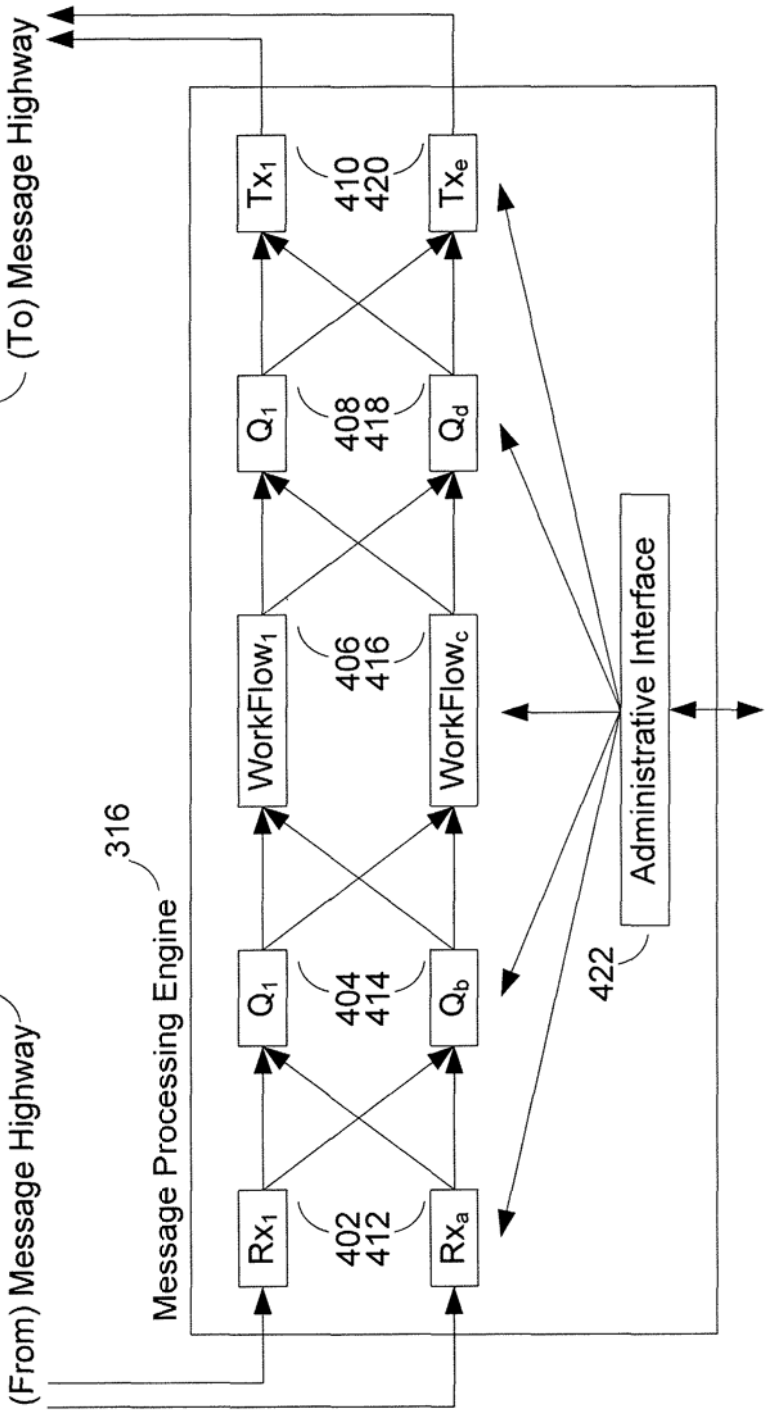
400
310
(To) Message Highway
316
Message Processing Engine
310
(From) Message Highway
Rx1
Rxa
Q1
Qb
WorkFlow1
WorkFlowc
Q1
Qd
Tx1
Txe
402
412
404
414
406
416
408
418
410
420
Administrative Interface
422

FIG. 4

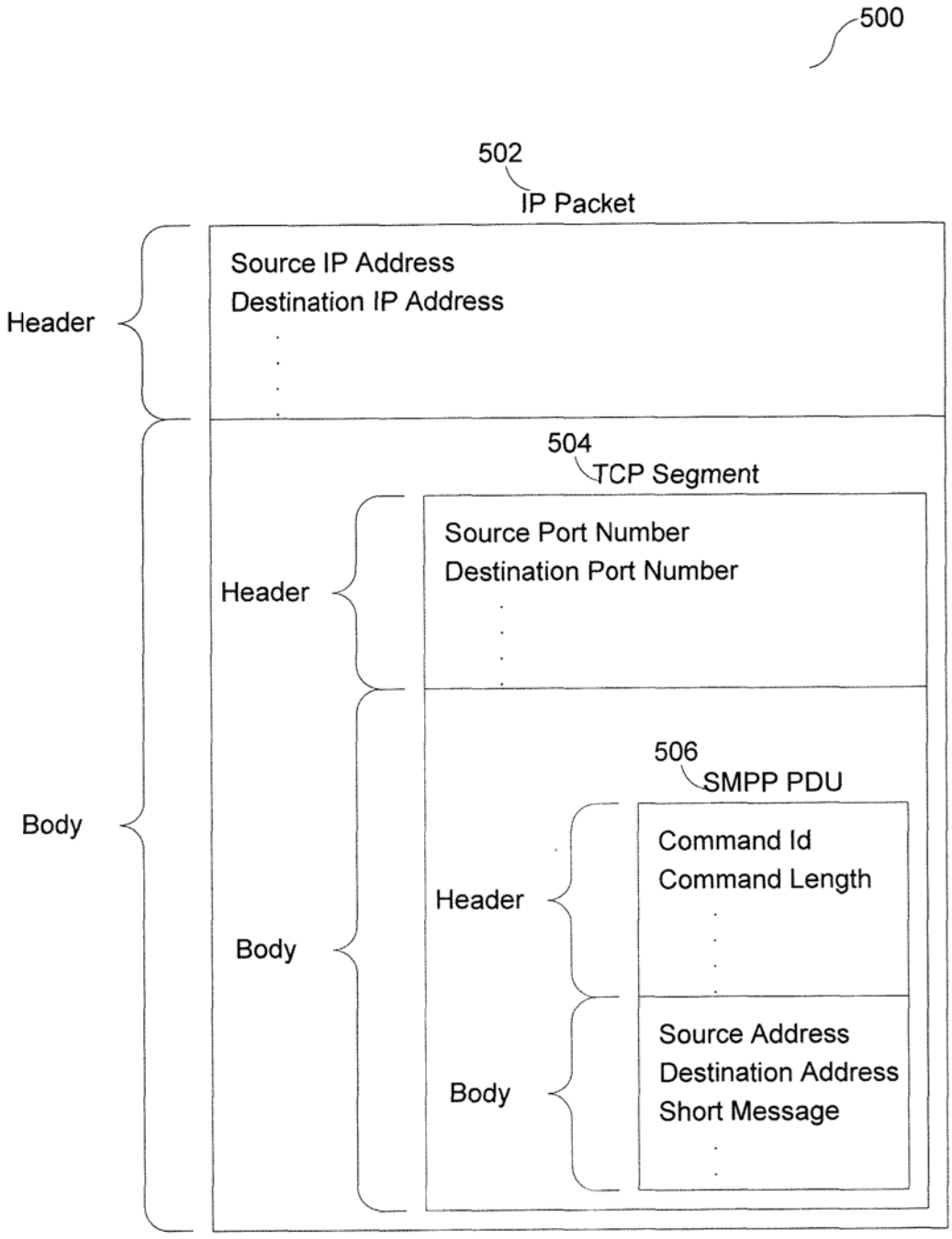
500
502
IP Packet
Source IP Address
Destination IP Address
Header
504
TCP Segment
Source Port Number
Destination Port Number
Header
506
SMPP PDU
Command Id
Command Length
Header
Body
Source Address
Destination Address
Short Message
Body
Body

FIG. 5

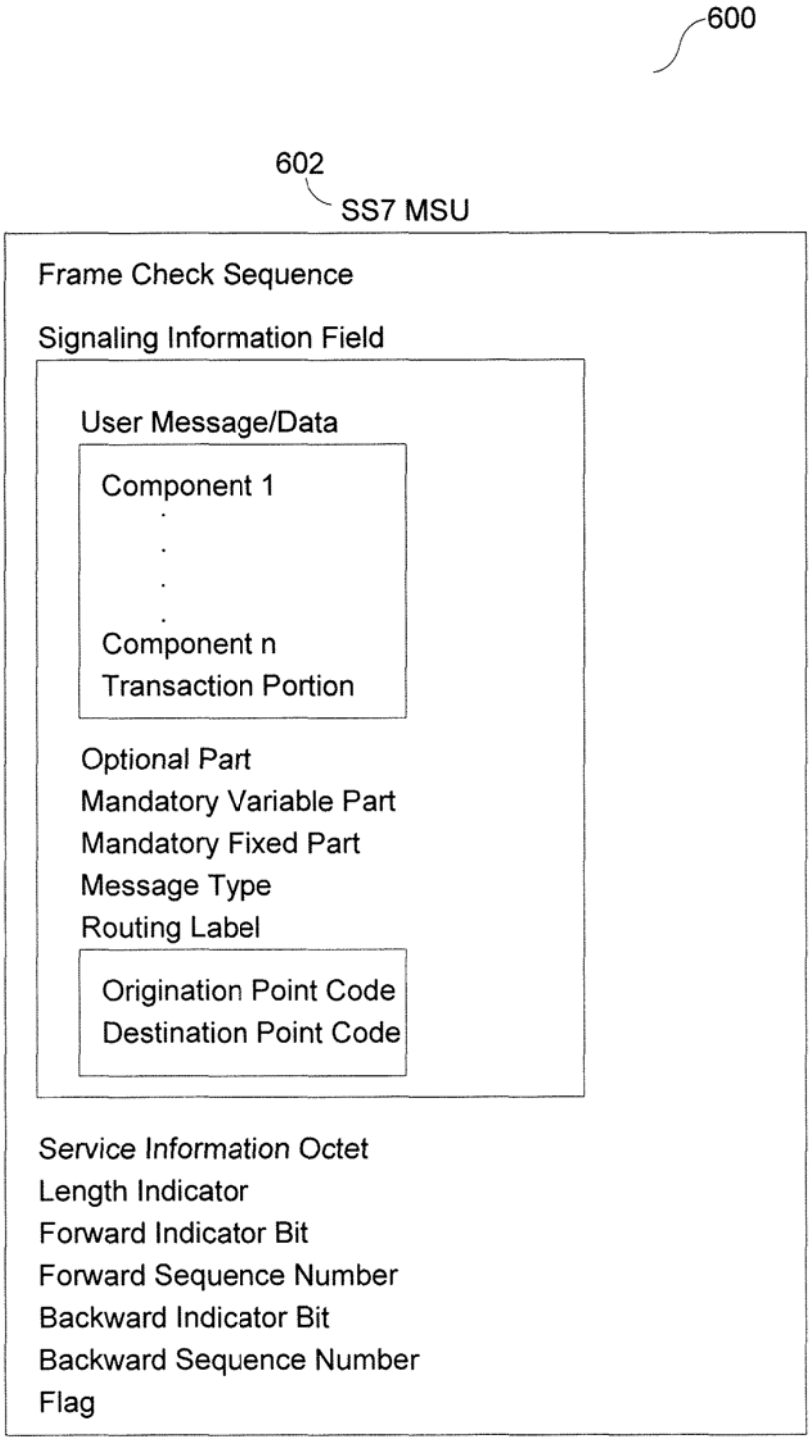
600
602
SS7 MSU
Frame Check Sequence
Signaling Information Field
User Message/Data
Component 1
Component n
Transaction Portion
Optional Part
Mandatory Variable Part
Mandatory Fixed Part
Message Type
Routing Label
Origination Point Code
Destination Point Code
Service Information Octet
Length Indicator
Forward Indicator Bit
Forward Sequence Number
Backward Indicator Bit
Backward Sequence Number
Flag

FIG. 6

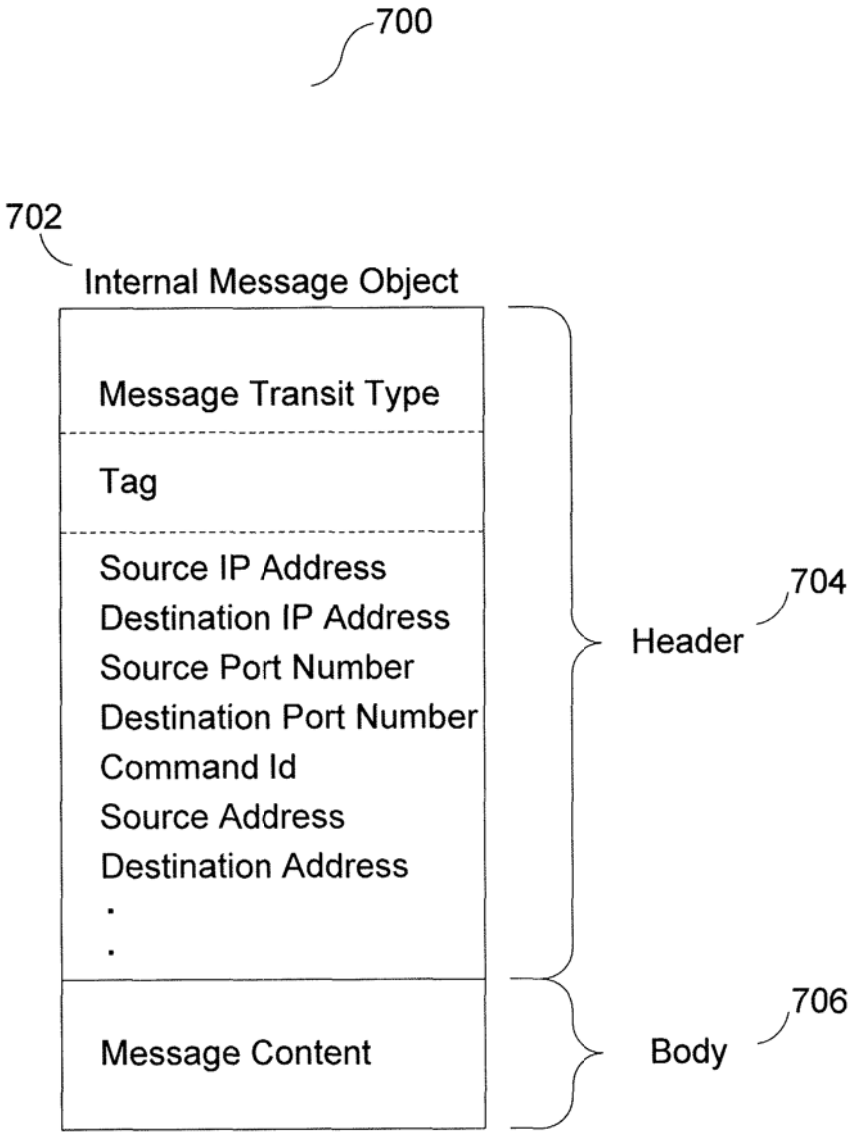
700
702
Internal Message Object
Message Transit Type
Tag
Source IP Address
Destination IP Address
Source Port Number
Destination Port Number
Command Id
Source Address
Destination Address
.
.
704
Header
Message Content
706
Body

FIG. 7

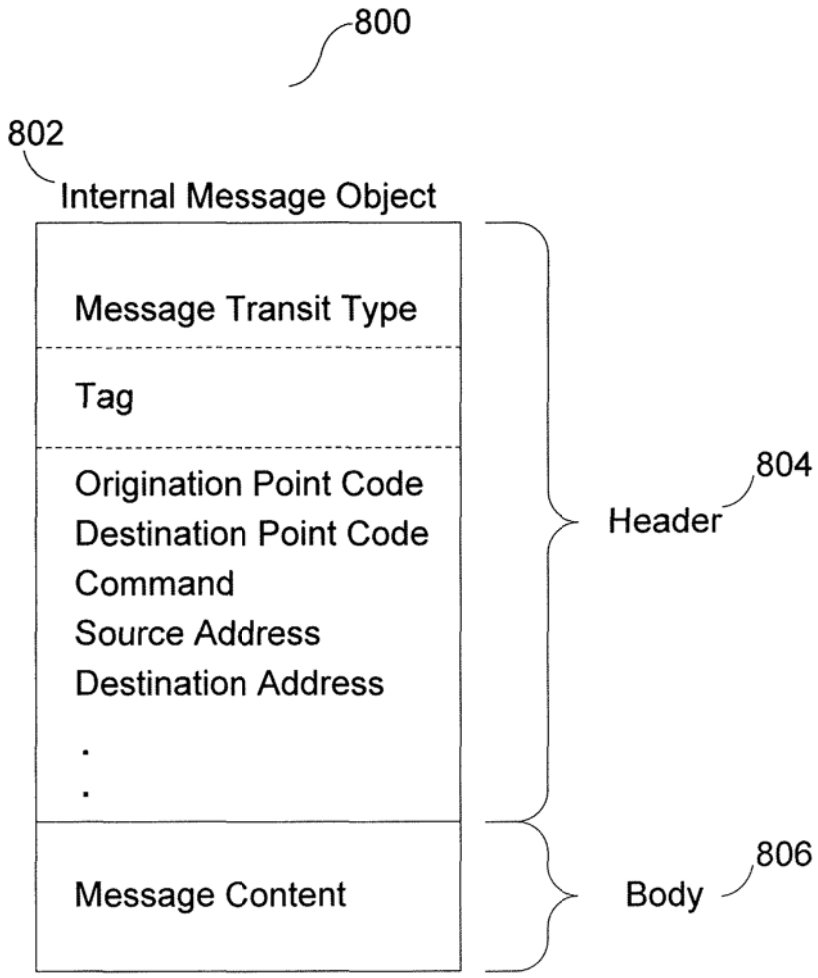
800
802
Internal Message Object
Message Transit Type
Tag
Origination Point Code
Destination Point Code
Command
Source Address
Destination Address
.
.
Message Content
804
Header
806
Body

FIG. 8

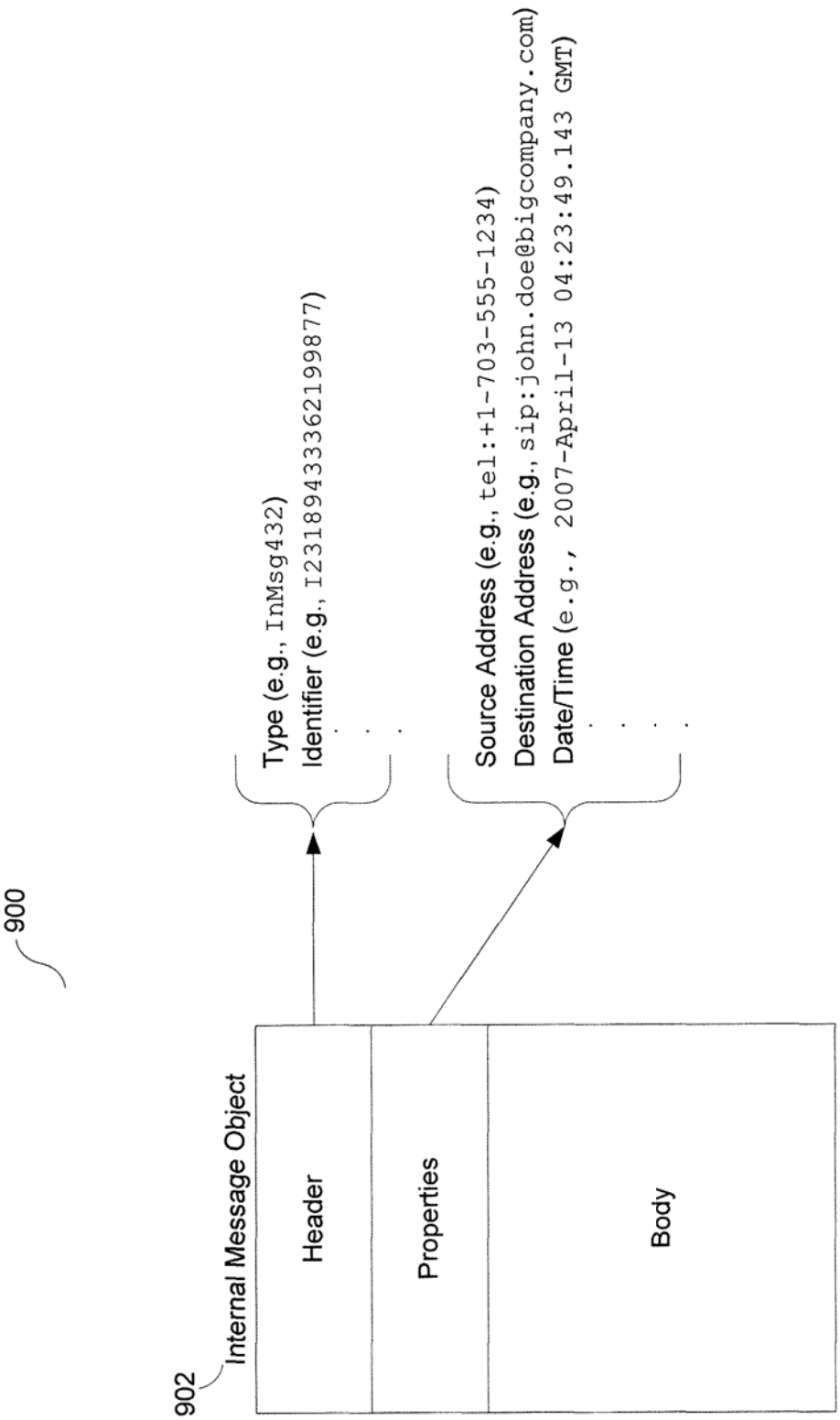
900
902
Internal Message Object
Header
Properties
Body
Type (e.g., InMsg432)
Identifier (e.g., I231894333621998 77)
Source Address (e.g., tel:+1-703-555-1234)
Destination Address (e.g., sip:john.doe@bigcompany.com)
Date/Time (e.g., 2007-April-13 04:23:49.143 GMT)

FIG. 9

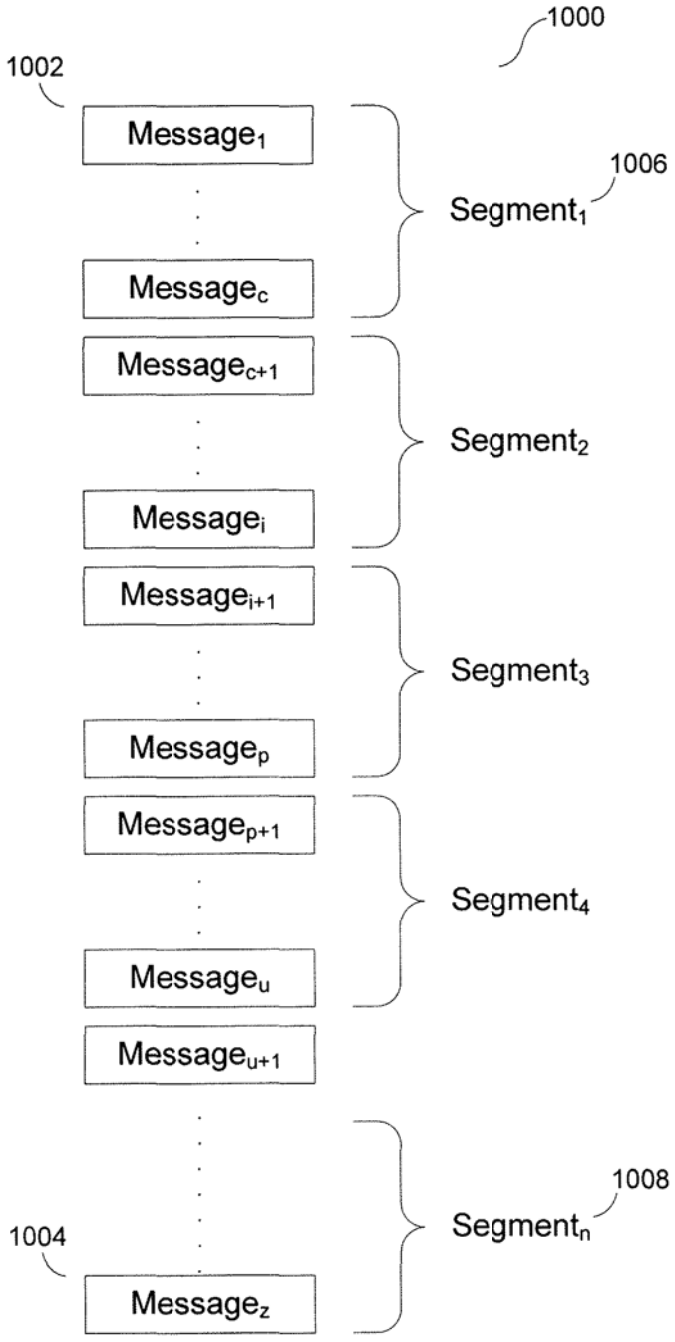
1000
1002
Message1
Messagec
Segment1
1006
Messagec+1
Messagei
Segment2
Messagei+1
Messagep
Segment3
Messagep+1
Messageu
Segment4
Messageu+1
1004
Messagez
Segmentn
1008

FIG. 10

1100

Segment	Number of Messages
1	435
2	435
3	435
4	435
Total	**1740**

FIG. 11a

1102

Segment	Number of Messages
1	290
2	290
3	290
4	290
5	290
6	290
Total	**1740**

FIG. 11b

1200

Segment	Number of Messages
1	265
2	415
3	390
4	150
Total	**1220**

FIG. 12a

1202

Segment	Number of Messages
1	163
2	235
3	143
4	216
5	92
6	170
7	201
Total	**1220**

FIG. 12b

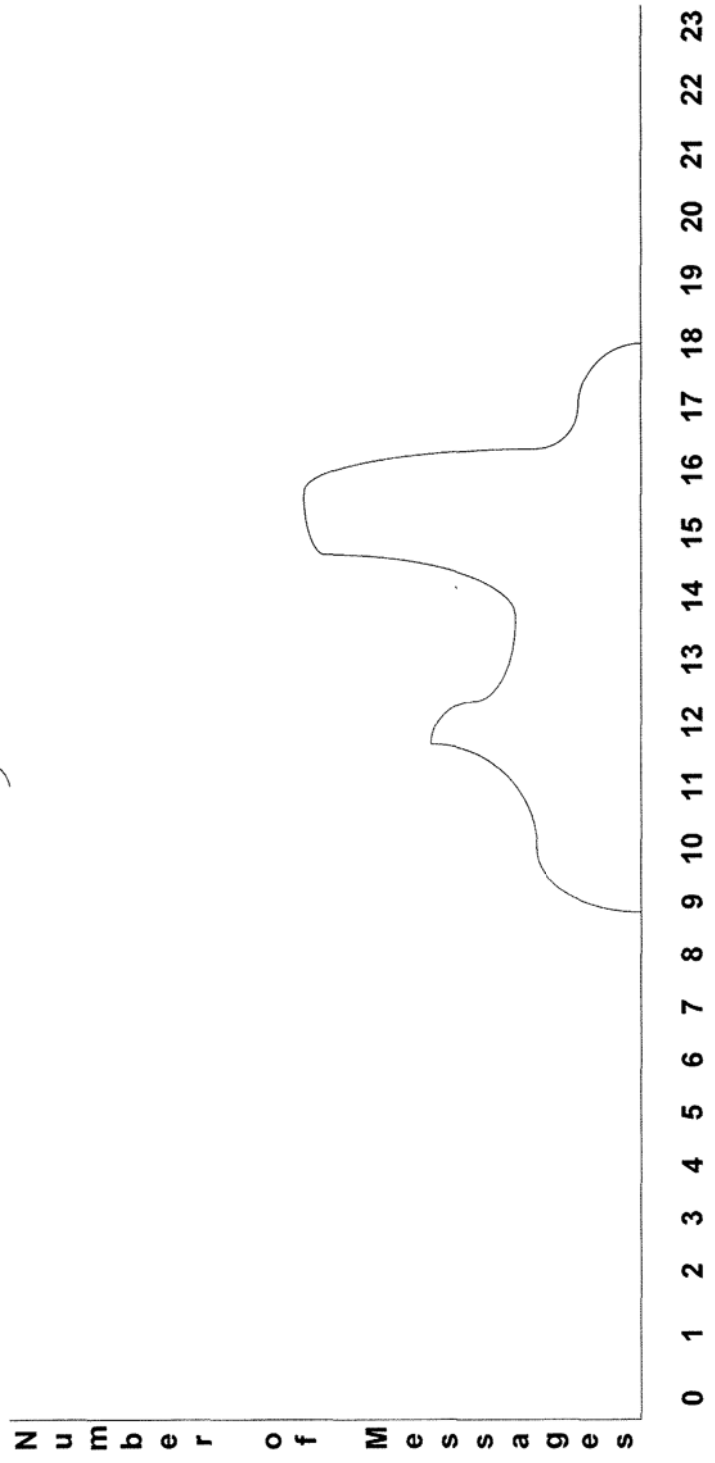
1300
Number of Messages
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Time of Day

FIG. 13

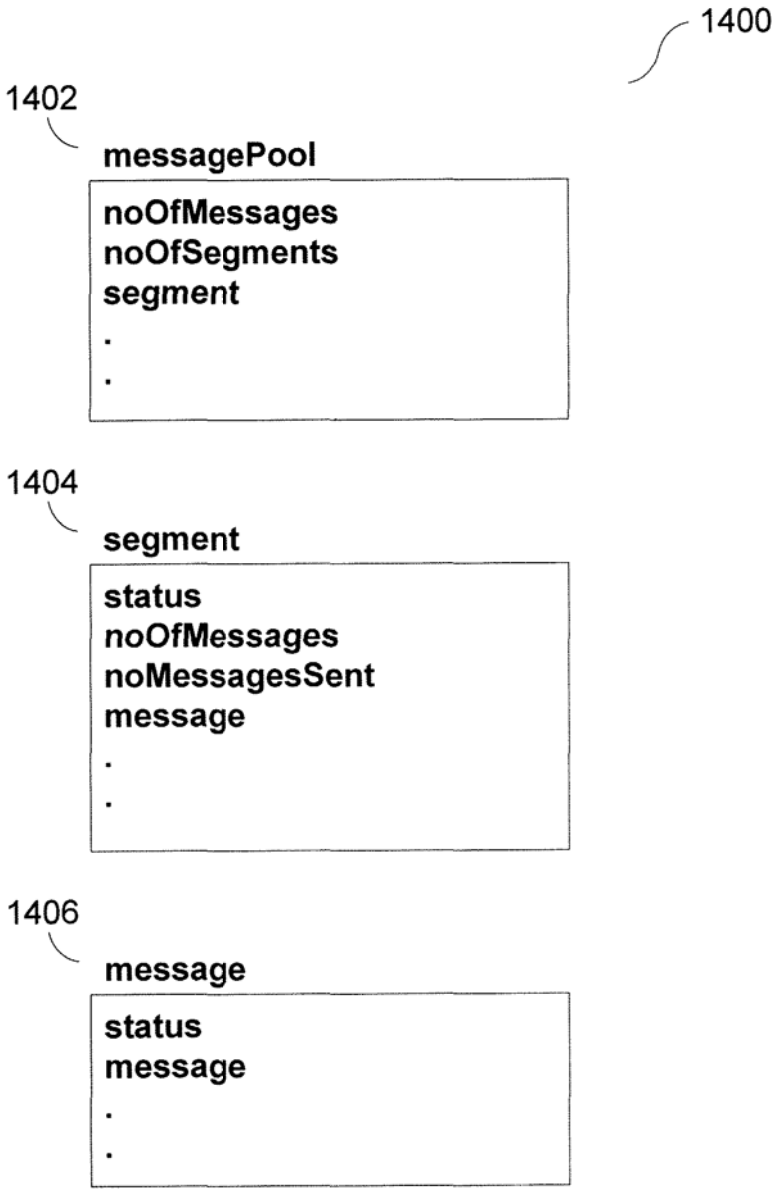
1400
1402
messagePool
noOfMessages
noOfSegments
segment
.
.
1404
segment
status
noOfMessages
noMessagesSent
message
.
.
1406
message
status
message
.
.

FIG. 14

1500

1502

```
ProcessPool
{
   /* Perform any initialization activities */
   Initialize ();

   /* Process each segment in the message pool */
   for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
   {
      /* Process each message in segment segmentNo */
      for (messageNo = 1; messageNo <= messagePool.segment [segmentNo].noOfMessages; messageNo ++)
      {
         /* Send message messageNo in segment segmentNo */
         send message messagePool.segment [segmentNo].message [messageNo].message;

         /* Optionally wait for and optionally process message send ACK */

         /* Wait (after message is sent) */
         pause (time1);
      }

      /* Wait (after segment is done) */
      pause (time2);
   }

   /* Perform any cleanup activities */
   Finalize ();
}
```

FIG. 15

1600

1602

```
ProcessPool
{
   /* Perform any initialization activities */
   Initialize ();

   /* Spin through all of the segments in the message pool */
   for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
   {
      /* A segment starts out as closed with no messages sent */
      messagePool.segment [segmentNo].status = CLOSED;
      messagePool.segment [segmentNo].noOfMessagesSent = 0;
      for (messageNo = 1; messageNo <= messagePool.segment [segmentNo].noofMessages;
           messageNo ++)
         messagePool.segment [segmentNo].message [messageNo].status = UNSENT;
   }

   /* Start processing */
   notDone = TRUE;
   noOfChildTreads = 0;
   while (notDone)
   {
      /* Is the window closed? */
      if (window.status == CLOSED)
      {
         /* Yes, stop any sending activity ... */
         for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
           if (messagePool.segment [segmentNo].status == WORKING)
           {
              Cleanly terminate child thread with ProcessSegment (segmentNo);
              noOfChildThreads --;
              messagePool.segment [segmentNo].status = CLOSED;
           }

         /* ... and loop (wait for the window to open) */
         continue;
      }
```

Continued on Figure 16b

FIG. 16a

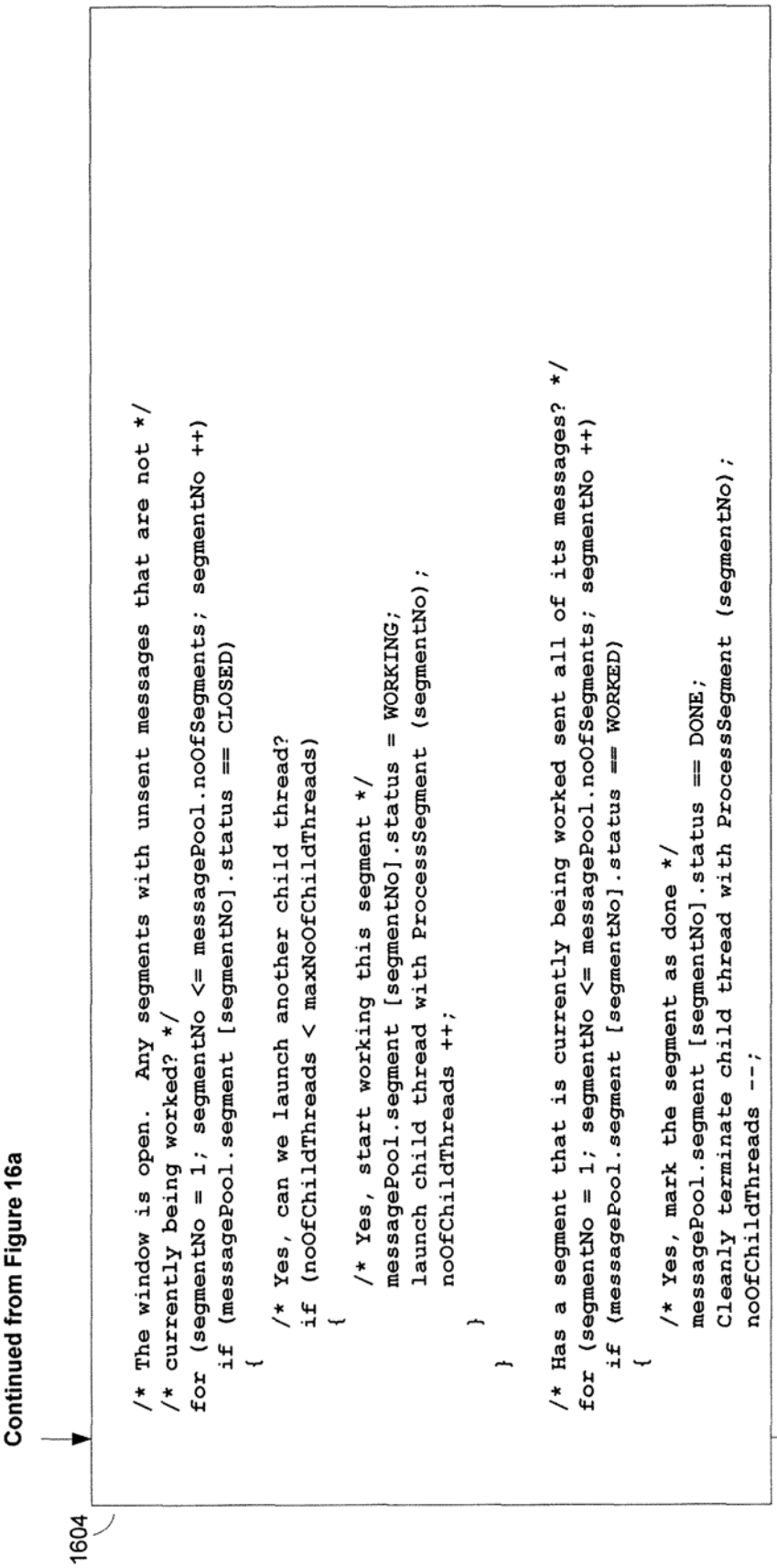
Continued from Figure 16a
1604
/* The window is open. Any segments with unsent messages that are not */
/* currently being worked? */
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
if (messagePool.segment [segmentNo].status == CLOSED)
{
/* Yes, can we launch another child thread?
if (noOfChildThreads < maxNoOfChildThreads)
{
/* Yes, start working this segment */
messagePool.segment [segmentNo].status = WORKING;
launch child thread with ProcessSegment (segmentNo);
noOfChildThreads ++;
}
}
/* Has a segment that is currently being worked sent all of its messages? */
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
if (messagePool.segment [segmentNo].status == WORKED)
{
/* Yes, mark the segment as done */
messagePool.segment [segmentNo].status == DONE;
Cleanly terminate child thread with ProcessSegment (segmentNo);
noOfChildThreads --;
Continued on Figure 16c

FIG. 16b

Continued from Figure 16b

1606

```
/* Has a segment that is currently being worked reached a messages-sent */
/* threshold such that we can now open another segment? */
foundSegment = FALSE;
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
   if (messagePool.segment [segmentNo].status == WORKING)
   {
      if ( (messagePool.segment [segmentNo].noOfMessagesSent /
            messagePool.segment [segmentNo].noOfMessages) >= messagesSentThreshold)
      {
         /* Yes, start working another segment */
         for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
            if (messagePool.segment [segmentNo].status == CLOSED)
            {
               messagePool.segment [segmentNo].status = WORKING;
               launch child thread with ProcessSegment (segmentNo);
               noOfChildThreads ++;

               foundSegment = TRUE;
               break;
            }
      }

   if (foundSegment == TRUE)
     break;
   }

/* Are we all done? */
noOfSegmentsDone = 0;
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo ++)
   if (messagePool.segment [segmentNo].status == DONE)
      noOfSegmentsDone ++;
if (noOfSegmentsDone == messagePool.noOfSegments)
{
   /* Yes */
   notDone = FALSE;
}
```

Continued on Figure 16d

FIG. 16c

Continued from Figure 16c

1608

```
      pause (time5);
   }

   /* Perform any cleanup activities */
   Finalize ();
}
ProcessSegment (segmentNo)
{
   /* Process each message in segment segmentNo */
   for (messageNo = 1; messageNo <= messagePool.segment (segmentNo).noOfMessages; messageNo ++)
   {
      /* Send message messageNo in segment segmentNo */
      send message messagePool.segment [segmentNo].message [messageNo].message;

      /* Optionally wait for and optionally process message send ACK */

      messagePool.segment [segmentNo].message [messageNo].status = SENT;
      messagePool.segment [segmentNo].noOfMessagesSent ++;

      /* Wait (after message is sent) */
      pause (time3);
   }

   messagePool.segment [segmentNo].status = WORKED;
   /* Wait (after segment is done */
   pause (time4);

   /* Exit */
}
```

FIG. 16d