

(19) World Intellectual Property Organization  
International Bureau



(43) International Publication Date  
10 January 2008 (10.01.2008)

PCT

(10) International Publication Number  
**WO 2008/005447 A2**

(51) International Patent Classification:  
*H04L 29/06* (2006.01)

(21) International Application Number:

PCT/US2007/015377

(22) International Filing Date: 2 July 2007 (02.07.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/818,763 6 July 2006 (06.07.2006) US

11/823,237 27 June 2007 (27.06.2007) US

(71) Applicant (for all designated States except US): **HONEYWELL INTERNATIONAL INC.** [US/US]; 101 Columbia Road, P.O. Box 2245, Morristown, New Jersey 07962 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **ZHAO, Jianhua** [US/US]; 6 Tullamore Court, Ringoes, New Jersey 08551 (US). **GRANATELLI, David** [AU/AU]; 146 Lilyfield

Road, Lilyfield, New South Wales 2040 (AU). **KANJIL, Muslim, G.** [US/US]; 477 Wild Mint Lane, Allentown, Pennsylvania 18104 (US).

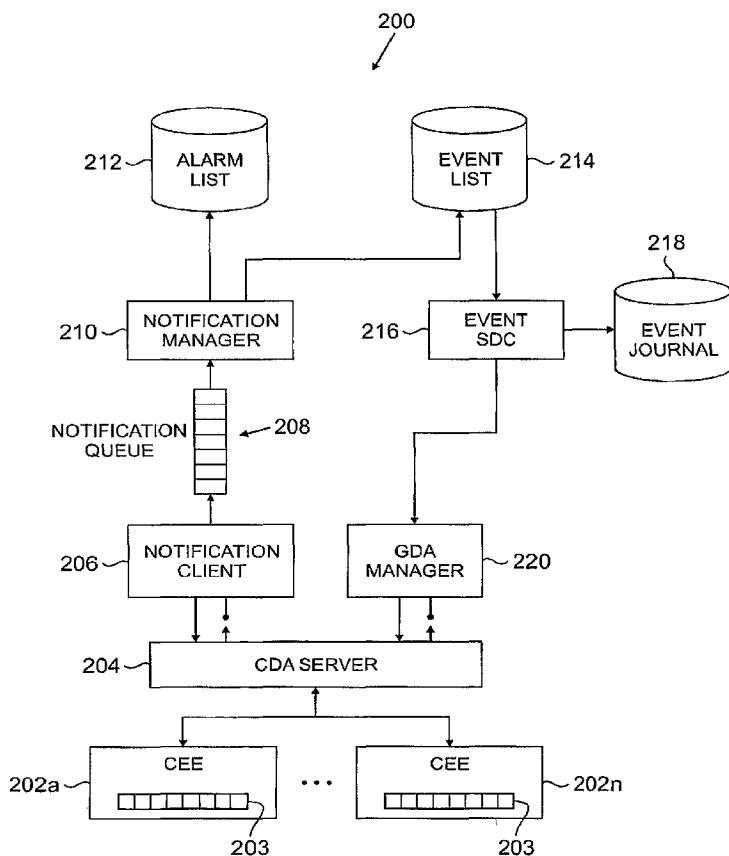
(74) Agents: **MIOLOGOS, Anthony** et al.; Honeywell International Inc., 101 Columbia Road, P.O. Box 2245, Morristown, New Jersey 07962 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM,

[Continued on next page]

(54) Title: APPARATUS AND METHOD FOR GUARANTEED BATCH EVENT DELIVERY IN A PROCESS CONTROL SYSTEM



(57) Abstract: A controller (104a-104b) controls one or more process elements (102a-102b) in a process control system (100). The controller (104a-104b) also generates and buffers multiple events associated with operation of the controller (104a-104b). Each event is associated with a unique identifier, such as a numerical identifier. A supervisory device (106a-106b, 216) receives at least some of the events from the controller (104a-104b). The supervisory device (106a-106b, 216) determines if any of the events have not been received from the controller (104a-104b) based on the unique identifiers. The supervisory device (106a-106b, 216) also initiates an event recovery if at least one of the events has not been received from the controller (104a-104b). During the event recovery, the controller (104a-104b) recovers at least one event, such as any event having a numerical identifier greater than a numerical identifier identified by the supervisory device (106a-106b, 216). The controller (104a-104b) communicates the at least one recovered event to the supervisory device (106a-106b, 216) during the event recovery.



ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

**Published:**

- *without international search report and to be republished upon receipt of that report*

APPARATUS AND METHOD FOR GUARANTEED BATCH EVENT DELIVERY  
IN A PROCESS CONTROL SYSTEM

5

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority under 35 U.S.C.  
§ 119(e) to U.S. Provisional Patent Application No.  
60/818,763 filed on July 6, 2006, which is hereby  
10 incorporated by reference.

[0002] This application is related to U.S. Patent  
Application No. 11/453,119 filed on June 14, 2006, which is  
hereby incorporated by reference.

15

TECHNICAL FIELD

[0003] This disclosure relates generally to control  
systems and more specifically to an apparatus and method  
for guaranteed batch event delivery in a process control  
20 system.

## BACKGROUND

[0004] Processing facilities are often managed using process control systems. Example processing facilities include chemical, pharmaceutical, paper, and petrochemical production plants. Among other operations, process control systems typically interact with and control industrial equipment in the processing facilities, such as equipment used to produce chemical, pharmaceutical, paper, or petrochemical products. Often times, these or other types of products are manufactured or processed in batches.

[0005] Regulated industries often require complete batch records of all aspects related to the manufacture of batch products. In fact, the data records for batch execution are often as important as the products themselves. Non-regulated industries, though not having the same requirements for batch record completeness, also typically use batch records to support functions such as quality assurance analyses. As a result, complete or near-complete batch records are often desirable or required in a process control system.

## SUMMARY

[0006] This disclosure provides an apparatus and method for guaranteed batch event delivery in a process control system.

5       [0007] In a first embodiment, a method includes generating multiple events associated with operation of a controller in a process control system. The method also includes storing the events in a buffer and communicating the events to a supervisory device associated with the  
10 controller. The method further includes receiving a request to recover one or more of the events. In addition, the method includes recovering at least one of the events from the buffer and communicating the at least one recovered event to the supervisory device.

15       [0008] In particular embodiments, each event is associated with a unique identifier, and the request identifies the unique identifier associated with one of the events.

20       [0009] In other particular embodiments, the unique identifier identified in the request includes a numerical identifier. Also, recovering the at least one event from the buffer includes recovering any event associated with a numerical identifier that is greater than the numerical identifier identified in the request.

25       [0010] In still other particular embodiments, the method further includes generating an alarm in response to recovering less than all desired events from the buffer.

30       [0011] In a second embodiment, an apparatus includes at least one memory configured to store a unique identifier associated with a first event received from a controller in a process control system. The apparatus also includes at least one processor configured to receive a second event

from the controller and to determine whether any events between the first event and the second event have not been received from the controller using the unique identifier. The at least one processor is also configured to initiate an event recovery if at least one event between the first event and the second event has not been received from the controller.

[0012] In a third embodiment, a computer program is embodied on a computer readable medium. The computer program includes computer readable program code for generating multiple events associated with operation of a controller in a process control system. The computer program also includes computer readable program code for storing the events in a buffer and computer readable program code for communicating the events to a supervisory device associated with the controller. The computer program further includes computer readable program code for receiving a request to recover one or more events. In addition, the computer program includes computer readable program code for recovering at least one of the events from the buffer and computer readable program code for communicating the at least one recovered event to the supervisory device.

[0013] In a fourth embodiment, a system includes a controller configured to control one or more process elements in a process control system. The controller is also configured to generate and buffer multiple events associated with operation of the controller. Each event is associated with a unique identifier. The system also includes a supervisory device configured to receive at least some of the events from the controller and to determine if any of the events have not been received from the controller based on the unique identifiers. The

supervisory device is also configured to initiate an event recovery by the controller if at least one of the events has not been received from the controller. The controller is further configured to recover at least one event and to  
5 communicate the at least one recovered event to the supervisory device during the event recovery.

[0014] Other technical features may be readily apparent to one skilled in the art from the following figures, descriptions, and claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0015] For a more complete understanding of this disclosure, reference is now made to the following description, taken in conjunction with the accompanying  
5 drawings, in which:

[0016] FIGURE 1 illustrates an example process control system supporting guaranteed batch event delivery;

[0017] FIGURE 2 illustrates additional details of an example process control system supporting guaranteed batch  
10 event delivery;

[0018] FIGURES 3A through 3D illustrate example data flows in a process control system supporting guaranteed batch event delivery;

[0019] FIGURE 4 illustrates example parallel data flows  
15 in a process control system supporting guaranteed batch event delivery;

[0020] FIGURES 5 and 6 illustrate a specific example of a portion of a process control system supporting guaranteed batch event delivery;

[0021] FIGURE 7 illustrates an example user interface  
20 supporting guaranteed batch event delivery in a process control system; and

[0022] FIGURE 8 illustrates an example method for  
25 guaranteed batch event delivery in a process control system.



## DETAILED DESCRIPTION

[0023] FIGURE 1 illustrates an example process control system 100 supporting guaranteed batch event delivery. The embodiment of the process control system 100 shown in  
5 FIGURE 1 is for illustration only. Other embodiments of the process control system 100 may be used without departing from the scope of this disclosure.

[0024] In this example embodiment, the process control system 100 includes one or more process elements 102a-102b.  
10 The process elements 102a-102b represent components in a process or production system that may perform any of a wide variety of functions. For example, the process elements 102a-102b could represent equipment used to manufacture chemical, pharmaceutical, paper, or petrochemical products.  
15 Each of the process elements 102a-102b includes any hardware, software, firmware, or combination thereof for performing one or more functions in a process or production system.

[0025] Two controllers 104a-104b are coupled to the  
20 process elements 102a-102b. The controllers 104a-104b control the operation of the process elements 102a-102b. For example, the controllers 104a-104b could be capable of providing control signals to the process elements 102a-102b for controlling the production of chemical, pharmaceutical,  
25 paper, or petrochemical products. Each of the controllers 104a-104b includes any hardware, software, firmware, or combination thereof for controlling one or more of the process elements 102a-102b. The controllers 104a-104b could, for example, include one or more processors 105 and  
30 one or more memories 107 storing instructions and data used, generated, or collected by the processor(s) 105. As particular examples, the processors 105 could include

processors of the POWERPC processor family running the GREEN HILLS INTEGRITY operating system or processors of the X86 processor family running a MICROSOFT WINDOWS operating system.

5       **[0026]** Two servers 106a-106b are coupled to the controllers 104a-104b. The servers 106a-106b perform various functions to support the operation and control of the controllers 104a-104b and the process elements 102a-102b. For example, the servers 106a-106b could log  
10 information collected or generated by the controllers 104a-104b, such as status information related to the operation of the process elements 102a-102b. The servers 106a-106b could also execute applications that control the operation of the controllers 104a-104b, thereby controlling the  
15 operation of the process elements 102a-102b. In addition, the servers 106a-106b could provide secure access to the controllers 104a-104b. Each of the servers 106a-106b includes any hardware, software, firmware, or combination thereof for providing access to or control of the  
20 controllers 104a-104b. Each of the servers 106a-106b could, for example, include one or more processors 109 and one or more memories 111 storing instructions and data used, generated, or collected by the processor(s) 109 (such as software executed by the servers 106a-106b). As  
25 particular examples, the processors 109 could include processors of the POWERPC processor family running the GREEN HILLS INTEGRITY operating system or processors of the X86 processor family running a MICROSOFT WINDOWS operating system.

30       **[0027]** One or more operator stations 108a-108b are coupled to the servers 106a-106b, and one or more operator stations 108c are coupled to the controllers 104a-104b. The operator stations 108a-108b represent computing or

communication devices providing user access to the servers 106a-106b, which could then provide user access to the controllers 104a-104b and the process elements 102a-102b. The operator stations 108c represent computing or communication devices providing direct user access to the controllers 104a-104b. As particular examples, the operator stations 108a-108c could allow users to review the operational history of the process elements 102a-102b using information collected by the controllers 104a-104b and/or the servers 106a-106b. The operator stations 108a-108c could also allow the users to adjust the operation of the process elements 102a-102b, controllers 104a-104b, or servers 106a-106b. Each of the operator stations 108a-108c includes any hardware, software, firmware, or combination thereof for supporting user access and control of the system 100. Each of the operator stations 108a-108c could, for example, include one or more processors 113 and one or more memories 115 storing instructions and data used, generated, or collected by the processor(s) 113 (such as software executed by the operator stations 108a-108c). In particular embodiments, the operator stations 108a-108c could represent personal computers executing a MICROSOFT WINDOWS operating system.

[0028] In this example, at least one of the operator stations 108b is remote from the servers 106a-106b. The remote station is coupled to the servers 106a-106b through a network 110. The network 110 facilitates communication between various components in the system 100. For example, the network 110 may communicate Internet Protocol (IP) packets, frame relay frames, Asynchronous Transfer Mode (ATM) cells, or other information between network addresses. The network 110 may include one or more local area networks (LANs), metropolitan area networks (MANs),

10

wide area networks (WANs), all or a portion of a global network such as the Internet, or any other communication system or systems at one or more locations.

[0029] In this example embodiment, the system 100 includes two additional servers 112a-112b. The servers 112a-112b execute various applications to control the overall operation of the system 100. For example, the system 100 could be used in a processing or production plant or other facility, and the servers 112a-112b could execute applications used to control the plant or other facility. As particular examples, the servers 112a-112b could execute applications such as enterprise resource planning (ERP), manufacturing execution system (MES), or any other or additional plant or process control applications. Each of the servers 112a-112b includes any hardware, software, firmware, or combination thereof for controlling the overall operation of the system 100.

[0030] As shown in FIGURE 1, the system 100 includes various redundant networks 114a-114b and single networks 116a-116c that support communication between components in the system 100. Each of these networks 114a-114b and 116a-116c represents any network or combination of networks facilitating communication between components in the system 100. The networks 114a-114b and 116a-116c could, for example, represent Ethernet networks.

[0031] In one aspect of operation, one or more of the controllers 104a-104b may generate batch execution events, which represent events that occur during controller operation in the process control system 100. The batch execution events generated by a controller 104a-104b are typically communicated to a supervisory server (such as one or more servers 106a-106b) for more permanent storage, such as in an event journal or other database. The batch

execution events generated by a controller 104a-104b are also buffered or temporarily stored within the controller's memory. This supports the ability of the supervisory server to recover some or all of the events generated by a controller, such as events generated during periods when communication between the controller and the supervisory server has been interrupted. To support this event recovery, the system 100 provides a mechanism for tracking events and for resending "lost" or "missed" events to the supervisory server after an abnormal situation is cleared and the system 100 is back in normal operation.

[0032] In some embodiments, as batch execution events are generated within a controller 104a-104b, the events are numbered or otherwise identified in sequence with a unique identifier. Each of the events may, for example, be associated with an 8-byte numerical value as the events are generated or saved in the controller's event buffer. These unique identifiers are sent to the supervisory server along with the events. Upon receipt, the supervisory server examines the sequence of unique identifiers associated with the received batch events to identify potential gaps. If a gap is detected in the sequence of unique identifiers, this indicates that at least one event was not received by the supervisory server. The supervisory server then requests that the controller resend the missing or lost event or events (such as a specified event, events within a range of unique identifiers, or all events starting at a specific unique identifier). This helps to support the recovery of missing or lost batch events, thereby helping to ensure collection of all or substantially all events by the supervisory server for storage in event journals or other appropriate destination(s). In addition, if an event is permanently lost (it cannot be recovered due to event

buffer overload in the controller or other reason), the system 100 may generate an alarm to notify a user.

[0033] In particular embodiments, when an event buffer in a controller 104a-104b is used, a user can configure the event buffer size. As particular examples, small, medium, and large event buffers could accommodate 120, 240, and 740 events, respectively. The user could select a proper buffer size based on any suitable criteria, such as the application in which the controller is used. Once the buffer size is selected, the controller 104a-104b may reserve the appropriate amount of space in its memory for the buffer.

[0034] Although FIGURE 1 illustrates one example of a process control system 100 supporting guaranteed batch event delivery, various changes may be made to FIGURE 1. For example, a process control system could include any number of process elements, controllers, servers, operator stations, and networks. Also, the makeup and arrangement of the process control system 100 is for illustration only. Components could be added, omitted, combined, or placed in any other configuration according to particular needs. Further, while described as being used to produce certain types of products, the process control system 100 could be used in any other manner. In addition, FIGURE 1 illustrates one operational environment in which guaranteed batch event delivery can be used. Guaranteed batch event delivery could be used in any other device or system.

[0035] FIGURE 2 illustrates additional details of an example process control system supporting guaranteed batch event delivery. In particular, FIGURE 2 illustrates additional details regarding a particular implementation of the controllers 104a-104b and servers 106a-106b in the process control system 100 of FIGURE 1. The additional

details of the process control system shown in FIGURE 2 are for illustration only. Other embodiments of the process control system 100 may be used without departing from the scope of this disclosure.

5       **[0036]** In this example, the process control system 100 includes one or more control execution environments 202a-202n. The control execution environments 202a-202n represent execution environments used or supported by the controllers 104a-104b in the process control system 100.  
10       The control execution environments 202a-202n may support a wide variety of functions, such as the scheduling and execution of various control and background tasks. One example of a control execution environment is provided in U.S. Patent Application No. 11/175,848, which is hereby  
15       incorporated by reference.

**[0037]** The control execution environments 202a-202n generate batch events, such as events related to the processing or production of a batch of a product. Regulated industries often require complete batch records  
20       of all aspects related to batch execution. The batch records can include the batch events captured as a part of the manufacturing of batch products. Non-regulated industries may not have the same requirements for complete batch records, but batch execution results may be captured  
25       for other reasons (such as quality assurance, continuous improvement, and process manufacturing optimization). As a result, it is often important or essential that all batch events generated by batch engines in the control execution environments 202a-202n be captured and stored in an event  
30       journal or other storage location(s). In this example embodiment, each of the control execution environments 202a-202n includes or supports an event buffer 203 for temporarily storing batch events generated by the control

execution environments 202a-202n. As described below, this helps to facilitate the retrieval or recovery of lost or missing events, helping to ensure that the batch records in the process control system 100 are entirely or substantially complete. Each buffer 203 represents any suitable structure for storing and facilitating retrieval of batch events.

[0038] Due to memory limitations within controller products (such as purpose-built hardened controller platforms), batch execution events are typically transmitted on a timely basis to a supervisory server. Also, due to the nature of controller/server communications, there are times when a link or one or more of these platforms are not available. This may occur, for example, during redundant server failover, redundant controller failover, or interruption of a communication link between the controller and the server. The batch events that are generated in a control execution environment are retained within the event buffer 203 in the controller and sent to the supervisory server when the defined interruption no longer exists. For instance, the supervisory server can request the retransmission of events that the supervisory server determines are lost or missing. The following describes how the recovery of events from the control execution environments 202a-202n can occur.

[0039] The control execution environments 202a-202n are in communication with a control data access (CDA) server 204. The CDA server 204 provides secure data access to and from the control execution environments 202a-202n. For example, the CDA server 204 may ensure that only authorized sources (such as a supervisory server) are able to access and interact with the control execution environments 202a-202n. The CDA server 204 includes any hardware, software,



firmware, or combination thereof for controlling data access.

[0040] A notification client 206 facilitates communication between the control execution environments 202a-202n (via the CDA server 204) and a notification queue 208. The notification client 206 may, for example, receive batch events from the control execution environments 202a-202n and provide the events to the notification queue 208 for storage. The notification client 206 includes any hardware, software, firmware, or combination thereof for providing access to the notification queue 208. The notification queue 208 includes any suitable structure for storing multiple events or other notifications.

[0041] The notifications from the notification queue 208 are retrieved by a notification manager 210. The notification manager 210 may, for example, retrieve batch events associated with the control execution environments 202a-202n from the notification queue 208. The notification manager 210 may then provide the notifications to one or more suitable destinations. For example, if a notification represents an alarm, the notification may be provided to an alarm list 212 for storage. If the notification represents an event, the notification may be stored in an event list 214. The notification manager 210 includes any hardware, software, firmware, or combination thereof for retrieving information from the notification queue 208. The alarm list 212 and the event list 214 represent any suitable structures for storing multiple notifications or other information.

[0042] Events from the event list 214 are retrieved by an event system data component (event SDC) 216, which stores the events in an event journal 218. The event SDC 216 also determines, using unique identifiers associated

with the events, whether any events are missing or have been lost. For example, the unique identifiers could represent a sequence of numbers, and the event SDC 216 may identify missing events by identifying gaps in the unique  
5 identifiers associated with the received events. When the event SDC 216 needs to recover one or more lost or missing events from the control execution environments 202a-202n, the event SDC 216 may communicate with the notification client 206 or a generic data access (GDA) manager 220  
10 (depending on the circumstances). Either of these components can request transmission or retransmission of one or more buffered events by the control execution environments 202a-202n. The event SDC 216 includes any hardware, software, firmware, or combination thereof for  
15 journaling events and identifying missing events. The event journal 218 includes any suitable structure for storing or journaling events, such as an SQL database. The GDA manager 220 includes any hardware, software, firmware, or combination thereof for providing access to the control  
20 execution environments 202a-202n.

**[0043]** In particular embodiments, the various components shown in FIGURE 2 are divided between one or more of the controllers 104a-104b and one or more of the servers 106a-106b. For example, the control execution environments  
25 202a-202n could reside in or be associated with multiple controllers, while the remaining components could reside in or be supported by a supervisory server.

**[0044]** In one aspect of operation, events may be buffered in the control execution environments 202a-202n,  
30 and the events can be associated with unique identifiers. A supervisory server (via the event SDC 216) can identify missing or lost events based on the unique identifiers associated with the events. If necessary, lost or missing

events can be retransmitted from the control execution environments 202a-202n to the supervisory server.

[0045] Although FIGURE 2 illustrates additional details of one example of a process control system supporting guaranteed batch event delivery, various changes may be made to FIGURE 2. For example, the makeup and arrangement of the components in FIGURE 2 are for illustration only. Components could be added, omitted, combined, or placed in any other configuration according to particular needs.

[0046] FIGURES 3A through 3D illustrate example data flows in a process control system supporting guaranteed batch event delivery. In particular, FIGURES 3A through 3D illustrate example data flows between the components of the process control system 100 that are shown in FIGURE 2. The embodiments of the data flows shown in FIGURES 3A through 3D are for illustration only. Other data flows may be used without departing from the scope of this disclosure.

[0047] The data flow in FIGURE 3A may be used when a supervisory server is first starting up, meaning when the supervisory server is becoming active. In this case, the supervisory server may have missed events previously communicated by one or more controllers during the server's startup period. In this situation, the following sequence of events may occur.

(1) At server start-up, the notification client 206 subscribes to and receives a notification from the CDA server 204.

(2) The CDA server 204 obtains a list of control execution environments (CEE) 202a-202n that are capable of generating events, such as by using information from a system repository. The CDA server 204 returns the list of control execution environments to the notification client 206, such as by using a callback function.

(3) The notification client 206 queries the event SDC 216 for the sequence identifiers of the last events received by the event SDC 216 from the identified control execution environments.

5       (4) The event SDC 216 queries the event journal 218 if the event SDC 216 cannot resolve the notification client's query with its internal list.

          (5) If necessary, the event journal 218 returns the list of last-received sequence identifiers from the  
10       identified control execution environments to the event SDC 216.

          (6) The event SDC 216 passes this information to the notification client 206.

          (7) The notification client 206 passes this  
15       information to the CDA server 204.

          (8) The CDA server 204 requests an event recovery from each control execution environment by specifying, for each control execution environment, the appropriate sequence identifier received from the event SDC 216.

20       (9) Each control execution environment publishes its events starting at the sequence identifier requested by the CDA server 204.

          (10) For each control execution environment, the CDA server 204 generates an event recovery begin bracket for  
25       that control execution environment and sends the begin bracket to the notification client 206. The begin bracket contains the sequence identifier of the corresponding request for event recovery and the sequence identifier of the first recovered event. The recovery bracket is  
30       followed by a chain of one or more recovered events.

          (11) For each control execution environment, the recovered events are provided to the event SDC 216, which updates its internal list with the sequence identifier of

the recovery begin bracket.

(12) The recovered events are journaled or logged into the event journal 218.

[0048] The data flow in FIGURE 3B may be used when the  
5 controller restarts and reconnects to the supervisory  
server or when a cold reload of the controller occurs.  
During this time, the controller may be out of contact with  
its supervisory server, and some events may be generated.  
In this situation, the following sequence of events may  
10 occur.

(1) A control execution environment 202a-202n notifies the CDA server 204 that the controller has restarted.

(2) The CDA server 204 returns its code for the control execution environment with the execution  
15 environment's load time to the notification client 206, such as by using a callback function.

(3) The notification client 206 queries the event SDC 216 for the sequence identifier of the last event received by the event SDC 216 from the control execution  
20 environment.

(4) The event SDC 216 queries the event journal 218 if the event SDC 216 cannot resolve the notification client's query with its internal list.

(5) If necessary, the event journal 218 returns the  
25 last-received sequence identifier that was received from the control execution environment to the event SDC 216.

(6) The event SDC 216 passes this information to the notification client 206.

(7) The notification client 206 passes this  
30 information to the CDA server 204.

(8) The CDA server 204 requests an event recovery from the control execution environment by specifying the sequence identifier received from the event SDC 216.

(9) The control execution environment publishes its events starting at the sequence identifier requested by the CDA server 204.

(10) The CDA server 204 generates an event recovery  
5 begin bracket for the control execution environment and sends it to the notification client 206. The recovery bracket is followed by a chain of one or more recovered events.

(11) The recovered events are provided to the event  
10 SDC 216, which updates its internal list with the sequence identifier of the recovery begin bracket.

(12) The recovered events are journaled or logged into the event journal 218.

**[0049]** The data flow in FIGURE 3C may be used when one  
15 or more events are lost. This could be caused by various reasons, such as a server failover, a controller failover, a network disruption, or excessive server load. In this situation, the following sequence of events may occur.

(1) One or more events are lost (for whatever reason),  
20 meaning one or more events transmitted from a control execution environment are not received or journaled properly by the event SDC 216.

(2) An event with an unexpected sequence identifier arrives at the event SDC 216. This event may have, for  
25 example, a numerical identifier that does not immediately follow the numerical identifier of the previously received event.

(3) The event SDC 216 commands an event recovery through the GDA manager 220 when it detects this  
30 occurrence.

(4) The CDA server 204 receives the recovery command.

(5) The control execution environment continues sending events that are not expected by the event SDC 216

(events that do not have the one or more missing numerical identifiers identified by the event SDC 216).

(6) The CDA server 204 requests an event recovery from the control execution environment and specifies the sequence identifier at which the recovery should begin.

(7) The unexpected events propagate to the notification client 206.

(8) The unexpected events reach the event SDC 216. The event SDC 216 may not update its internal list after receiving these unexpected events, and it may not re-command another recovery (at this point).

(9) The unexpected events are journaled or logged into the event journal 218.

(10) The control execution environment publishes events starting at the sequence identifier requested by the CDA server 204.

(11) The CDA server 204 generates an event recovery begin bracket for the notification client 206, followed by a chain of one or more recovered events.

(12) The recovered events reach the event SDC 216, which updates its internal list with the sequence identifier of the recovery begin bracket.

(13) The events are journaled or logged into the event journal 218.

**[0050]** The data flow in FIGURE 3D may be used when a control execution environment completes execution of a control recipe, which (as explained below) defines the production requirements for a specific product and is used to produce a single batch of the product. In this situation, the following sequence of events may occur.

(1) Upon completion of execution of a control recipe, the control recipe reports a recipe complete event to the CDA server 204.

(2) The CDA server 204 sends the recipe complete event to the notification client 206.

(3) The recipe complete event reaches the event SDC 216.

5       (4) The recipe complete event is journaled or logged into the event journal 218.

(5) If all recipe events have been journaled (the recipe complete event is in sequence and no events with unexpected sequence identifiers have been received), the event SDC 216 notifies the CDA server 204 via the GDA manager 220 that all recipe events for the control recipe have been journaled.

(6) The CDA server 204 receives the signal. At this point, the control recipe can now be safely removed.

15       **[0051]** Although FIGURES 3A through 3D illustrate examples of data flows in a process control system supporting guaranteed batch event delivery, various changes may be made to FIGURES 3A through 3D. For example, the events and other information may flow in any other suitable manner in the process control system. Also, other data flows could be used in other or additional situations in the process control system to support guaranteed batch event delivery.

25       **[0052]** FIGURE 4 illustrates example parallel data flows in a process control system supporting guaranteed batch event delivery. In particular, FIGURE 4 illustrates example data flows between the components of the process control system 100 that are shown in FIGURE 2. The embodiments of the data flows shown in FIGURE 4 are for illustration only. Other data flows may be used without departing from the scope of this disclosure.

30       **[0053]** In this example, the solid lines represent the normal data flow for notifications such as alarms,



sequences of events (SOE), live events, and messages. These notifications generally flow from a control execution environment 202a-202n through the CDA server 204, the notification client 206, and the notification manager 210. Live events are archived in the event list 214 and are journaled by the event SDC 216. However, the number of certain types of events, such as events from control recipes, may be quite large depending on the application. As a result, the number of events may overwhelm existing alarm and system event generation mechanisms in process control systems.

**[0054]** To support the use of guaranteed batch event delivery, the process control system could include a separate data path (represented by the dashed lines) for batch events. This separate data path includes a batch event processor 402, which handles the processing of batch events and the insertion of the batch events into the event list 214. In this example, batch events are sent from the notification client 206 to the batch event processor 402 for handling, rather than to the notification manager 210. This allows certain batch events to bypass the notification manager 210 and be sent to the event list 214 directly, allowing the event SDC 216 to retrieve and journal the events without requiring the use of the notification manager 210.

**[0055]** Although FIGURE 4 illustrates one example of parallel data flows in a process control system supporting guaranteed batch event delivery, various changes may be made to FIGURE 4. For example, while shown as including a path for batch events and a path for all other notifications, the process control system could support any number of data paths for the notifications. Also, parallel data flows may not be required in a process control system,

and the batch event processor 402 could be omitted.

[0056] FIGURES 5 and 6 illustrate a specific example of a portion of a process control system supporting guaranteed batch event delivery. In particular, FIGURE 5 illustrates an example process control system 500, and FIGURE 6 illustrates an example mechanism supporting guaranteed batch event delivery in the process control system 500. The embodiment of the process control system supporting guaranteed batch event delivery shown in FIGURES 5 and 6 is for illustration only. Other embodiments of the process control system could be used without departing from the scope of this disclosure.

[0057] The process control system 500 shown in FIGURE 5 could be implemented in any suitable system, such as by using the various components shown in FIGURES 1 and 2. As shown in FIGURE 5, the process control system 500 includes one or more unit control modules (UCMs) 502, recipe control modules (RCMs) 504, sequential control modules (SCMs) 506, and control modules (CMs) 508. In some embodiments, the unit control modules 502, recipe control modules 504, sequential control modules 506, and control modules 508 are distributed across multiple control execution environments in the process control system 100, such as the controllers 104a-104b.

[0058] A unit control module 502 generally represents or is associated with a process unit 510 that contains one or multiple pieces of processing equipment, where use of the process unit 510 occurs after acquisition of the unit control module 502. As described in U.S. Patent Application No. 11/453,119, requester function blocks and resource function blocks can be invoked by, incorporated into, or otherwise used by the various control modules 502-508. The resource function blocks represent objects that

can be acquired and released by the requester function blocks, where the requester function blocks use arbitration requests to attempt to acquire the resource function blocks. In these embodiments, a unit control module 502  
5 can be acquired by a recipe control module 504, which allows the process unit 510 to be used during execution of the recipe control module 504.

[0059] A recipe control module 504 generally represents information defining the production requirements for one or  
10 more specific products (or parts thereof), where execution of a recipe control module 504 could result in the production of a single batch of the one or more products (or parts thereof). A recipe control module 504 could include a header, a procedure, a formula, and any equipment  
15 requirements. The procedure in a recipe control module 504 is defined by a set of phases represented by phase function blocks 512. Each phase of a recipe control module 504 is associated with a sequential control module 506, which interacts with one or more control modules 508 to implement  
20 one of the phases of the recipe control module 504. The control modules 508 provide access to and control over the actual process unit 510. The procedure in a recipe control module 504 could also include a set of step, transition, and synchronization blocks. Step blocks provide read/write  
25 access to the control modules 508, and synchronization blocks allow parallel execution of phase function blocks 512 or step blocks.

[0060] A recipe control module 504 manipulates the sequential control modules 506 through its phases, where  
30 the phases control the sequential control modules 506, monitor the execution states of the sequential control modules 506, and optionally propagate the execution states to the recipe control module 504. Phase function blocks

512 may also monitor their parent recipe control modules' states and propagate the states to their underlying sequential control modules 506 when their parent recipe control modules 504 enter abnormal states. In particular  
5 embodiments, the various control modules 504-508 could operate as defined by the International Electrotechnical Commission (IEC) 61131 and 61512 (including 61512-1) standards or in U.S. Patent No. 6,317,638 (all of which are hereby incorporated by reference).

10       **[0061]** When a recipe control module 504 is created, it may optionally be associated with a specific unit control module 502. The unit control module 502 can be acquired when the recipe control module 504 is executed and can optionally be released by the recipe control module 504 at  
15 any time, such as when the recipe control module 504 reaches a terminal state or at any other previous time. Various parameters can be defined within the phase function blocks 512. These parameters may include a reference to a sequential control module 506 for execution control, a  
20 resource name to be allocated when a phase is started, and a flag indicating if the acquired resource will be released at the end of the execution of a sequential control module 506. By default, the resource name may be based on the selected sequential or recipe control module's name. If  
25 there is no selected sequential control module 506, a phase can be used for resource management purposes. Once in a terminal state, all resources acquired by the recipe control module 504 may or may not be released depending on the configuration flag.

30       **[0062]** In some embodiments, the unit control modules 502, sequential control modules 506, and control modules 508 may represent common resources that can be shared between multiple recipe control modules 504. As a

particular example, a recipe control module 504 may generally need to acquire a unit control module 502 to execute its procedure. The recipe control module 504 could therefore acquire and release the unit control module 502  
5 as needed. As another particular example, recipe control modules 504 can acquire the sequential control modules 506 and the control modules 508 they need in order to implement the production of a product.

[0063] An owner pointer 514 in the unit control module  
10 502 identifies the recipe control module 504 currently being executed against the unit control module 502 (the recipe control module 504 that currently owns or has acquired the unit control module 502). An arbitration queue 516 identifies a specified number of recipe control  
15 modules 504 waiting to acquire the unit control module 502 to execute. When the current recipe control module 504 (identified by the owner pointer 514) releases the unit control module 502, the unit control module 502 can select the next recipe control module 504 from the queue 516 using  
20 any arbitration technique supported by the unit control module 502 or defined by the user (such as first-in, first-out or other technique). Similarly, the sequential control module 506 may include an arbitration queue 518. One or more recipe control modules 504 that are waiting to acquire  
25 the sequential control module 506 on behalf of its phase blocks are identified in the arbitration queue 518 of the sequential control module 506. The recipe control modules can be selected from the queue 518 in any order.

[0064] In general, a phase function block 512 in a  
30 recipe control module 504 represents a function block used to acquire, initiate execution of, and monitor execution of a sequential control module 506 or another recipe control module 504. For example, if a phase function block 512 is

configured to execute a sequential control module 506, the phase function block 512 may acquire the sequential control module 506, load formula parameters into the sequential control module 506, and start the sequential control module 506. The formula parameters represent a set of parameters used by a phase to communicate appropriate recipe data to a sequential control module 506, such as data controlling how the process unit 510 is used during the phase execution.

[0065] While a sequential control module 506 is executing, the phase function block 512 may monitor the status of the sequential control module 506. The phase function block 512 may also project data from the sequential control module 506 so that SCM execution can be monitored by a recipe control module 504 through the phase function block 512. Further, the phase function block 512 may command the sequential control module 506 to upload various report parameters to the phase function block 512, such as when execution of the sequential control module 506 is complete or another terminal state is reached. Depending on the configuration, a phase function block 512 may or may not wait for the sequential control module 506 to complete before the phase completes, which allows the recipe control module 504 to proceed to the following phase or step. Moreover, depending on the configuration, the phase function block 512 may or may not release any acquired resources (such as the sequential control module 506) at the completion of the phase. In addition, the recipe control modules 504, phase function blocks 512, or other modules may generate and report batch events.

[0066] The process control system 500 may be used with the guaranteed batch event delivery mechanisms described above. For example, the following types of events could be generated in the process control system 500. In some

embodiments, events can be generated at the control execution environment level, the sequential control module/recipe control module (SCM/RCM) level, and the SCM/RCM step and phase level. Event journaling can be  
5 enabled or disabled at each level, and enabling event journaling allows the reporting of specified events to the event SDC 216.

[0067] At the control execution environment level, a user may define how much memory is allocated for buffering  
10 the events (the size of the event buffer 203). The user can also enable or disable the journaling of all batch events or batch events for selected categories. The categories of events can include RCM/SCM execution, execution details, formula/recipe parameters, recipe  
15 headers, recipe resource allocations, and report/history parameters. The SCM events that can be added to the journal could include SCM/RCM execution starts, SCM/RCM execution termination states, SCM/RCM step starts, SCM/RCM step completes, RCM phase starts, RCM phase completes, SCM  
20 recipe parameter downloads (such as one event per parameter), SCM history parameter uploads (such as one event per parameter), RCM formula parameter downloads (such as one event per parameter), and RCM report parameter uploads (such as one event per parameter). An overall  
25 SCM/RCM event enable parameter can turn the entire SCM/RCM event reporting on or off (by default it can be set to "on" for RCM and to "off" for SCM).

[0068] The process control system 500 could provide for guaranteed batch event delivery using the mechanism shown  
30 in FIGURE 6. In particular, the mechanism shown in FIGURE 6 can be implemented within a control execution environment in a controller 104a-104b. This mechanism allows batch events to be buffered in the control execution environment.

It may also prevent certain user actions or other actions, such as preventing the release or deletion of a recipe control module 504 before all events associated with that recipe control module 504 have been journaled and an  
5 acknowledgement has been received from the event SDC 216 (as explained with reference to FIGURE 3D above).

[0069] As shown in FIGURE 6, a function block (such as a sequential control module 506 or a recipe control module 504) is executed in the control execution environment.  
10 During its execution, the function block generates a new event, which is defined by a class denoted clsBatchEvent. The function block reports the new event using a ReportNotice() function to an alarming block, which results in a ReportBlockNotice() function being executed. The  
15 block is then tagged, meaning the event gets reported using a ReportNotification() function. A block manager makes a determination as to whether the event represents a batch event. If not, the notification is provided to the notification client 206 using a class clsCiNotifClient.  
20 Otherwise, an event manager is notified of the batch event. The event manager is associated with a class clsBatchEventManager, which could represent a cardinality 1 class used to process batch events and manage event recovery. The functionalities of this class may include  
25 the creation and maintenance of a buffer (such as a buffer 203) for storing event information, the generation of batch events, and the regeneration or recovery of lost or missing events. This class could also support interfaces for the addition of event information and maintain the sequence  
30 identifiers for the events. The event manager provides batch events with their associated sequence identifiers to the notification client 206.

[0070] In this example, a clsBatchEventManager object



can be created during control execution environment startup. On creation of the clsBatchEventManager object, the sequence identifier for batch events in the control execution environment may be set to one or some other  
5 initial value.

[0071] In some embodiments, the memory for the buffer 203 may not be allocated during the creation of the clsBatchEventManager object since the size of the buffer 203 may not be fixed. Rather, the buffer size may be  
10 configured by a user, and the configuration data for the buffer 203 may not be available at the time that the clsBatchEventManager object is created. For example, the user may be provided with multiple options for the buffer size, such as none, small, medium, and large. These  
15 options could be provided in a control execution environment configuration form or in any other suitable manner. Once this parameter is selected and stored, a function in the clsBatchEventManager object may be invoked to create the memory required for the buffer 203. If the  
20 "none" option is selected, no memory is allocated for the buffer 203. Also, if memory is not available for the buffer 203, an exception can be returned.

[0072] In particular embodiments, the clsBatchEventManager object may be executed every base  
25 cycle of the control execution environment. Execution of this object may, for example, be part of the control execution environment's budgeted tasks. The only processing that the clsBatchEventManager object may perform in its execution time may be adding those batch events that  
30 were not previously added to a buffer in the CDA server 204. The number of events reported to the CDA server 204 every cycle may be limited (such as to five). The clsBatchEventManager object may provide an interface for

adding event information to the CDA buffer to support this functionality.

[0073] When the supervisory server initiates an event recovery, it can send a request to the control execution environment with the sequence identifier of an event, 5 indicating that this event was the last event received in sequence. This command is processed by the block manager, which could represent an unbudgeted task in the control execution environment. A method identifier may be used to 10 distinguish the event recovery request from other requests (such as a parameter store). Upon processing the event recovery request, the block manager may invoke the clsBatchEventManager object to reset its index to a new index containing the event with the sequence identifier 15 requested by the supervisory server. If event information to be recovered has already been overwritten in the buffer 203 of the control execution environment, an "event lost" alarm may be generated.

[0074] If the user selects a buffer size of "none," this 20 indicates that non-guaranteed batch event delivery is used with the control execution environment. When the buffer size is none, no memory may be allocated for the buffer 203, and a batch event received from a recipe control module 504 or a sequential control module 506 is directly 25 sent to the CDA server 204. To indicate that batch events are not guaranteed, a flag can be set in the CDA event structure to the appropriate value (such as false). A supervisory server, upon seeing the flag set to false, may not command event regeneration when events are missing.

30 [0075] The following represents specific details of a particular implementation of the guaranteed batch event delivery mechanism. These details are for illustration only. Other embodiments of a process control system that

supports guaranteed batch event delivery could be used.

[0076] In particular embodiments, the class representing batch events (clsBatchEvent) can include definitions of appropriate values. These values could include a sequence identifier (used to identify missing or lost events), a batch identifier (used to identify a batch of product being produced or processed), and a sub-condition field. The structure of an event notification may, for example, have the following parameters:

```

10      UINT64 SeqId ;           // Sequence number
      clsCID SubCondCID;       // CID for sub-condition
      // Data container that contains a text
      // string associated with the notification.
      DataContainerCreate<MAX_NOTIF_DC_BUFFER> StrValue;

```

15 The clsBatchEventManager object can include functions that are exposed to other classes. For example, the functions that are exposed to the block manager could include:

```

      BOOLEAN IF_AddEvent ( eventstructure ); // Add event
      Void IF_GenerateEvent ()                // Generate event
20      Void IF_EventRegenRequest (UINT64 SeqId)
          // Resend lost or missing event(s)
      void IF_AllocateBufferMemory (enumsize size)
          //Create buffer

```

25 The basic information required to regenerate a batch event can be stored in the event buffer 203 as a structure, such as a structure defined as:

```

      struct EventStructure
      {
30          DATETIME timestamp;
          IEEE_FLOAT64 Value;
          IEEE_FLOAT64 InfoField1;
          IEEE_FLOAT64 infoField2;
          UINT64 SequenceID;
35          UINT64 ExecutionId;
          clsIOID BlockIOID;
          NM_SUBTYPE EventType;
          INT16 infoField3;
      }
40

```

The size of this structure could be 60 bytes. To save memory, each field of the above structure can be used differently depending on the type of event being stored in the buffer 203.

[0077] There may also be multiple possible locations for the event buffer 203. For example, a single circular buffer could be maintained in the control execution environment, and all of the recipe control modules 504 and sequential control modules 506 assigned to the control execution environment may share the same buffer. The clsBatchEventManager object may be responsible for managing the buffer 203 and regenerating events in response to a recovery request. Since the buffer 203 is shared by all recipe control modules 504 and sequential control modules 506 in the control execution environment, it could be used more efficiently. For example, if some recipe control modules 504 or sequential control modules 506 do not have many events, the space in the buffer 203 can be used by the other recipe control modules 504 and sequential control modules 506 that have more events.

[0078] In other embodiments, each recipe control module 504 and each sequential control module 506 maintains its own circular buffer. In these embodiments, the control execution environment may only generate the unique sequence identifiers for the events.

[0079] The event buffer 203 could also be created in various ways. For example, the event buffer 203 could be created within a class as an array or in auxiliary memory. If created as a member of a class, unused or spare memory in a controller 104a-104b could be used for the buffer 203, such as 350KB of memory. If created in auxiliary memory, a user may have an option to decide the size of memory allocated for the events buffer 203.

[0080] In some embodiments, a batch creation event may contain a batch identifier, which could represent a string. It may be difficult to allocate storage space for strings associated with all elements of the buffer 203. Since a

batch creation event may not be generated as frequently as other events, only a few string objects may be allocated in a separate array called BatchIdBuffer. For batch creation events, a parameter in the event's structure may have an index to the batch identifier in the BatchIdBuffer array. The size of this buffer could be fixed, such as at 1/10 the size of the event buffer 203. Table 1 illustrates various possible sizes for the event buffer 203 and the associated BatchIdBuffer array.

Event Buffer Type	Capacity (Number of Events)	Size (in KB)	Capacity of BatchId Buffer	Size (in KB)	Total Size of buffers (in KB)
Small	120	7.03	12	0.5	7.53
Medium	240	14.065	24	1	15.06
Large	720	42.18	72	3.3	45.48

Table 1

A parameter of the enumeration type may be added to the definition file of a control execution environment for defining the size of the event buffer 203. Access to this parameter could be restricted, such as to developers only. This parameter could have a value of none, small, medium, or large (any of which could be the default). A SetEventBufferSize() function could store a value in this parameter. Upon a store of this parameter for a control execution environment, a call can be made to the clsBatchEventManager object, such as to an AllocateBufferMemory() function, in order to allocate the required memory for the buffer 203.

[0081] Once created, event information can be added to the buffer 203 via a call to an AddEvent() function. This call can be made in the block manager as part of the ReportNotification() function. In case the buffer 203 was not created (such as because of the user's selection), the events may not be reported. The AddEvent() function could be defined as follows:

36

```

clsCiNotifClient::pInstance's AddNotification function:
Void clsBatchEventManager::AddEvent
{strCkNotification * eventInfo)
{
5   If Buffer Allocated
   {
       if (LastElementIndex != MaxBufferSize)
       {
10          LastElementIndex++;
       }
       else
       {
           // indicating that buffer is full and the first
           // element added has to be overwritten
15          LastElementIndex = 0
       }

       struct EventStructure* BufferPtr = (struct
           EventStructure *)BufferAuxMem.GetPointer()
20          // extract the information from the strCkNotification
           // structure and store the value at
           // BufferPtr[LastElementIndex]
       }
   }
}

```

25     **[0082]** The batch events can be added to the CDA buffer  
by the clsBatchEventManager object. The  
clsBatchEventManager object may be executed as part of  
control execution environment's budgeted tasks. During its  
execution time, the event manager object may generate batch  
30 events using the information present in the buffer 203.  
Since this is part of the control execution environment's  
budgeted tasks, the processing time of the event manager  
object may be kept to a minimum so as to avoid cycle  
overrun, while at the same time being long enough to  
35 generate events at an acceptable rate so that the buffer  
203 does not overflow. With this consideration, the number  
of events generated in one execution cycle of the event  
manager object can be limited (such as to five). Since the  
event manager object can be executed periodically (such as  
40 every 50ms), it can still generate 100 events per second  
with this throttling. Another reason that this number may  
satisfy the needs of the process control system is that  
other components in the process control system could handle

a lesser amount of events, such as a maximum of 40 events per second. With this in mind, the clsBatchEventManager object could further be defined as follows:

```

Void clsBatchEventManager::EventGeneration()
5  {
    MaxEvenstPerExecution = 5
    counter = 0
    clsCiBatchEvent tempClsCiBatchEvt;
    if Event Buffer Allocated
10  {
        struct EventStructure* BufferPtr =
            (struct EventStructure *)BufferAuxMem.GetPointer()
        while (counter < MaxEvenstPerExecution And (there are
            events to be generated))
15  {
            // copy information from BufferPtr[CurrentEventIndex]
            // to tempClsCiBatchEvt depending on the event type.
            if (clsCiNotifClient::pInstance->
                AddNotification(tempClsCiBatchEvt)) != FALSE)
20  {
                If (CurrentEventIndex<MaxSize)
                    CurrentEventIndex ++ ;
                else
                    CurrentEventIndex = 0;
25  }
            else
            {
                // CDA buffer might be full retry in next cycle
                break;
30  }
            counter++
        }
    }
}

```

35     **[0083]** Event recovery can be commanded with a sequence identifier, which indicates that all events with sequence identifiers greater than the given sequence identifier should be regenerated. To handle an event recovery request, a method identifier can be used, and the event recovery can be processed as part of display request processing. In the case of event recovery, an interrupt mask register (IMR) may contain the sequence identifier. When the sequence identifier is 0, this indicates that the supervisory server is requesting regeneration of all events available in the buffer 203. A RegenEvents() function of the clsBatchEventManager object can be called with the

sequence identifier. The RegenEvents() function may reset a CurrentEventIndex parameter to point to the event structure containing the sequence identifier that is one greater than the received sequence identifier. The events  
 5 may then be regenerated during the next cycle of the event buffer's execution. With this in mind, the clsBatchEventManager object could further be defined as follows:

```

10      Void clsBatchEventManager::RegenEvents (UINT64 RecoverSeqID)
      {
          if Buffer created
          {
              struct EventStructure* BufferPtr = (struct EventStructure *)
              BufferAuxMem.GetPointer()
15              if(RecoverSeqID > 0)
                  {
                      RecoverEventIndex = GetIndex( SeqID);
                      // this function returns the index of the structure
                      // whose sequenceId = RecoverSeqId
20                  }
              else
              {
                  Recover EventIndex = First element in the buffer
              }
25              if (Recover EventIndex == -1)
              {
                  // this indicates that the event is not available in the
                  // buffer. Generate an alarm indicating loss of events.
                  RecoverEventIndex = GetNextIndex(SeqID)
30                  //this function returns index of the first structure
                  // having its sequenceId greater than RecoverSeqID.
              }
              CurentEventIndex = RecoverEventIndex;
35      }
  
```

In the next execution of the clsBatchEventManager object, events may be regenerated from the requested sequence identifier.

40     **[0084]** Static information associated with function blocks (such as recipe control modules 504 and sequential control modules 506) may not be stored in the event buffer 203. This information, such as block names, may be stored elsewhere, such as in a system repository. When a block is  
 45 deleted, this information may be lost, although the events



generated by the deleted block might still exist in the event buffer 203 after the block has been deleted. In case these events are regenerated, a static information field may contain either default values or error values. These  
5 may not be acceptable to the user. To avoid this, a recipe control module 504 or a sequential control module 506 may not be deleted until all of the events generated by the module have been journaled. For example, a parameter AllowBlockDelete may be added to a recipe control module  
10 504 or a sequential control module 506. If the module is not configured to generate events, the value of this parameter may be set to true. If the module is configured to generate events, the parameter is set to false. After the module has completed its execution, it generates a  
15 completion event. The supervisory server, on receiving the completion event, may initiate storage in the AllowBlockDelete parameter of the ExecutionID present in the complete event. If the ExecutionID matches the module's last run ExecutionID, the parameter  
20 AllowBlockDelete may be set to true. To delete a module, it may first be inactivated (such as when the module's state is idle). When the module is commanded to become inactive when it is in an idle state, the value AllowBlockDelete is checked. If AllowBlockDelete is true,  
25 the store of the ExecutionID in the parameter is allowed. If AllowBlockDelete is false, the store is not allowed to complete, the state of the module does not change, and a user is not allowed to change the module's state to inactive until all events generated by the module have been  
30 journaled (unless a "lost events" error is generated). If the supervisory server is offline, the store of the value in the AllowBlockDelete parameter may never occur, and the module might never be deleted. To overcome this, a

parameter AllowForceDelete (which could be a Boolean parameter) can be added to the modules. When inactivation of a module is commanded, the value of the AllowForceDelete parameter is checked. If this value is true, the state  
5 change is allowed irrespective of the value of the AllowBlockDelete parameter.

[0085] When the supervisory server commands an event recovery for a sequence identifier that is not present in the buffer 203, the clsBatchEventManager object may  
10 generate an alarm indicating a loss of events and may start regenerating the events starting with the first event having a sequence identifier greater than the one requested. The alarm may contain the sequence identifier(s) of the lost event(s).

15 [0086] The BatchID buffer for storing batch identifiers might also be very limited. The BatchID buffer may work in a manner similar to the main event buffer 203. During the addition of a new batch identifier, the new identifier may overwrite the oldest identifier when the BatchID buffer is  
20 full. It is possible that the BatchID buffer is overwritten but the corresponding event entry is still present in the event buffer 203. In this case, when a RegenEvents() function call for this event is received, the event can be generated with a BatchID field containing an  
25 empty string.

[0087] Since the event buffer 203 can be created in auxiliary memory, it may exist throughout the existence of the control execution environment. In case of a controller reload or deletion, all of the cardinality 1 objects can be  
30 recreated, including the clsBatchEventManager object. Therefore, no special processing may be required for a controller reload or control execution environment deletion. Also, in case of a RAM retention restart, an

41

auxiliary memory handle can be reconfirmed if memory had been allocated for the event buffer 203. Further, the auxiliary memory could undergo compaction, where data is moved in the auxiliary memory. When this occurs, the contents of the event buffer 203 need not be modified since the clsBatchEventManager object may not store memory pointers associated with the buffer 203 (so no pointers need to be updated in the buffer 203). Beyond that, no mechanism may be required for removing event information from the buffer 203. When implemented as a circular buffer, old event information is eventually overwritten with new event information. In addition, the clsBatchEventManager object may not be checkpointed (meaning batch events stored in the buffer 203 may be lost during a checkpoint restore), and the contents of the event buffer 203 in one controller can be tracked by a secondary or backup controller.

**[0088]** The block manager in FIGURE 6 could include functions for handling batch event processing. Since batch event processing may be common to all platforms, the changes may be made in a common file for the block manager. The functions in the block manager could include a function for periodic execution, such as the following function:

```
PeriodicExecution(void)
{
    ...
    if (isPrimary() != FALSE) // if controller is primary and
                             // not secondary controller
    {
        ...
        clsBatchEventManager::pInstance->
        GenerateEvents()
    }
}
```

The block manager could also include the ReportNotification() function, which is responsible for filling the CDA notification structure and calling the

AddNotification() function. When it receives a batch event, this function can fill the event structure and call the AddEvent() function in the clsBatchEventManager. This can be done using the following.

```

5      const BOOLEAN
      clsCkBlockManagerCommon::ReportNotification(strCkNotification*
      const CkNotif, const UINT32 TaggedBlockAlarmState)
      {
10          NM_TYPE NotificationType = ((strCkNotice *)
          (CkNotif->BlockNotice->Notice))->NotifClass;
          if (NotificationType != NM_TYPE_BATCH_EVENT &&
              NotificationType != NM_TYPE_PROCESS_ALERT )
          {
15              ...
          }
          else if (NotificationType == NM_TYPE_BATCH_EVENT)
          {
20              clsBatchEventManager::pInstance -> AddEvent(Event
              structure)
          }
      }

```

[0089] In addition, to support event recovery requests, a method identifier kBatchEvtRegen can be added to the enumerated method identifiers in the control execution environment. In case of an event recovery request, the clsBatchEventManager object's RegenEvents() function can be called with the sequence identifier present in the request. This can lead to the following.

```

30      BOOLEAN clsCkBlockManagerCommon::CkExecute (const enmDataAccess
      ListID, UINT32 NumIMRs)
      {
          ...
          while (pRequest != NULL)
35          {
              ...
              switch (MethodId)
              {
                  ...
40                  case kBatchEvtRegen:
                      get the requested SequenceId from CDA packet.
                      clsBatchEventManager::pInstance->
                      RegenEvents(RequestedSequenceID)
              }
45          }
      }

```

[0090] If the various components shown in FIGURE 5 (such as the recipe control modules 504 and the sequential

control modules 506) are used in the process control system 100 as shown in FIGURE 2, the components in FIGURE 2 could be implemented as follows.. These details represent one specific implementation of the process control system 100 as shown in FIGURE 2 and do not limit the scope of this disclosure.

[0091] An interface between the CDA server 204 and the notification client 206 may support the following special events to help support guaranteed event delivery: a recipe event recovery begin bracket and a control recipe complete event. These events may be uniquely identified by event sub-types enumerated in the control execution environment.

[0092] To support guaranteed event delivery, each of the recipe events may have a unique 8-byte increasing sequence identifier, assigned in the order that the events are generated. These events may flow from the control execution environment to the event SDC 216 before they are journaled. Throughout this data flow, events may be subject to loss, and the sequence identifiers are used to keep track of any lost events. Each recipe event may also carry the load time of its control execution environment, which can be used to record the execution environment cycle at which the recipe event is generated. The load time may record the time of the execution environment's most recent controller reload. The sequence identifier, together with the execution environment code and the execution environment's load time, may form a unique identifier for a recipe event in the whole process control system. These three combined fields may therefore be used for commanding a recipe event recovery on a particular control execution environment. The sequence identifier field could represent a UINT64 value, and the load time field may represent a FILETIME value.

[0093] Since the CDA server 204 may request a recipe event recovery from a control execution environment starting at the sequence identifier that is one greater than the requested sequence identifier received from the event SDC 216, the sequence identifier for a recipe event may not be assigned a value of zero. In terms of recipe event recovery, requesting a sequence identifier of zero could represent a request to recover all events starting at the oldest recipe event in the control execution environment. Also, when a control execution environment reloads, its load time may be reset to the current time, and its sequence identifier may be initiated to a value of one. Resetting the control execution environment's load time may ensure the current cycle of sequence identifiers is uniquely identified. Subsequent recipe events may then have an increasing sequence identifier with the same execution environment load time.

[0094] The following functions may be used to form at least part of the interface between the CDA server 204 and the notification client 206. A GNotificationSubscribe() function can be called when the notification client 206 subscribes for notification from the CDA server 204 on server startup. This function may have a callback function, which acts as the entry point to the notification client 206 for CDA notifications to be passed to the CDA server 204. This function may also include another callback function to the CDA server 204 so that the CDA server 204 can obtain a sequence identifier of the last journaled recipe event from each control execution environment. The GNotificationSubscribe() function can be implemented as follows.

```
// Type definition for Notification Callback Function
typedef int (*FPNotification)(NM_NOTIFICATION*);
// Type definition for GetSeqIDs Callback Function
```

45

```

typedef int (*FPGetSeqIDs)(WORD*, FILETIME*, UINT64**);
// Function for subscribing to notifications
CDAFUNC G_Status APIENTRY GNotificationSubscribe
(
5     // Callback through which notifications are passed to the
    // client
    FPNotification pfnNotifCallback
    // Callback through which CDA server gets the list of SeqIDs
    FPGetSeqIDs pfnGetSeqIDsCallback
10 )

```

[0095] A GRecipeEventRecovery() function may be used in the CDA server/notification client interface so a recipe event recovery from a control execution environment can be initiated when a loss of recipe events is detected. For example, the event SDC 216 may detect an out-of-order recipe event, such as an event having a sequence identifier higher than the expected sequence identifier for the execution environment. In this case, the event SDC 216 determines that one or more recipe events have been lost, and it calls this function through the GDA manager 220 to command a recipe event recovery for that execution environment. Supplied with this function are the execution environment's code, the execution environment's load time, and the sequence identifier of the recipe event at which the recovery begins. These three fields may uniquely identify the starting point for the recovery process. Upon receiving this command, the CDA server 204 requests a recipe event recovery from the execution environment starting at a sequence identifier that is one greater than the requested sequence identifier from the event SDC 216. The notification client 206 may receive the recovered recipe events through the same notification callback function. The first event in the recovery process may be the recipe event begin bracket, and it may be followed by one or more subsequent recipe events that have been recovered. The GRecipeEventRecovery() function could be defined as follows.

46

```

// Function to trigger a recipe event recovery from the execution
/ environment starting at the sequence ID requested
CDAFUNC G_Status APIENTRY GRecipeEventRecovery
(
5     const WORD NmEECode,
      const UINT64 NmSeqID,
      const FILETIME NmLoadTime
)

```

[0096] A GControlRecipeComplete() function provides the  
 10 supervisory server with the ability to notify the CDA  
 server 204 that it has journaled all recipe events for a  
 particular control recipe's execution. The event SDC 216  
 may call this function through the GDA manager 220 when it  
 receives a control recipe complete event and the event SDC  
 15 216 has journaled all recipe events for the control recipe.  
 This function may have a parameter that passes the batch  
 identifier of the completed control recipe to the CDA  
 server 204. The event SDC 216 may obtain the batch  
 identifier for the completed control recipe from the  
 20 appropriate field in the control recipe complete event.  
 The CDA server 204 may or may not need more information to  
 identify the control recipe (such as an execution  
 environment code or load time or a control module name).  
 The GControlRecipeComplete() function could be defined as  
 25 follows.

```

// Function to signal CDA server that all recipe events for the
// control recipe are journaled
CDAFUNC G_Status APIENTRY GControlRecipeComplete
(
30     const UINT32 NmBatchID
)

```

[0097] During event recovery, the recovered events are  
 preceded by a recipe event recovery begin bracket. The  
 recipe event recovery begin bracket may be sent when the  
 35 notification client 206 makes a notification subscription  
 or when the event SDC 216 commands a recipe event recovery  
 to the CDA server 204. This begin bracket, along with the  
 recovered recipe events, are originated from the control  
 execution environment. The begin bracket may be the first



event in the recovery process, and it marks the start of a sequence of one or more recovered recipe events to follow. The CDA server 204 may send this event to the notification client 206 through the usual notification mechanism. The notification client 206 may be the only recipient of this event, and it may not be distributed to console stations or other devices. A special sub-type may used to identify a recipe event recovery begin bracket. This sub-type may contain the sequence identifier requested by the event recovery and the sequence identifier of the first available recipe event recovered.

[0098] In normal circumstances, when the buffer 203 of recipe events in the control execution environment can service the recovery request, the returned sequence identifier may be one greater than the requested sequence identifier. However, if the buffer 203 in the control execution environment has overwritten the requested recipe event at which the event recovery was to begin, the execution environment may have to recover any remaining events starting with the oldest recipe event still in the buffer 203. In this case, the returned sequence identifier in the begin bracket would be the sequence identifier of the oldest recipe event recovered. Also, the control execution environment may generate a self-acknowledged alarm when it cannot recover all of the requested recipe events. The recipe event recovery may or may not require an end bracket. Additional details regarding the recipe event recovery begin bracket are provided in Table 2.

Field	Value
NmCategory	NC_BATCH_EVENT
NmType	NM_TYPE_BATCH_EVENT
NmSubType	NM_SUBTYPE_RECIPE_EVENT_RECOVERY
NmEECode	Execution environment code
NmPriority	NM_PRIORITY_JOURNAL
NmUnikId	Batch identifier

NmSeqId	Sequence identifier of the initial recipe event published for the recovery process (requested)
NmValue (overloaded)	Starting sequence identifier for which recipe events are commanded to recover (returned)
NmLoadTime	Load time of the execution environment
szAuxDesc	Description for this event, such as "Recipe event recovery for execution environment"

Table 2

The supervisory server may treat this recipe event recovery begin bracket as a journal-only event. It may therefore go into the event list 214, be displayed in an event summary, and may eventually reach the event SDC 216 and be journaled into the event journal 218.

[0099] The notification client 206 may check all received notifications to see whether it receives a recipe event recovery begin bracket. If it does, the notification client 206 may set a flag in the notification queue packet associated with the begin bracket to identify it. When this event reaches the event SDC 216, the event SDC 216 may identify the event using this flag and may update its internal list with the returned sequence identifier stored in the event's value field. This update may be done only if the sequence identifier in the in-memory list is less than the returned sequence identifier. This mechanism may be used to avoid the event SDC 216 from infinitely requesting recoveries for lost recipe events that have been wrapped-around in the control execution environment buffer 203.

[00100] A recipe event recovery begin bracket could be lost during transmission. To overcome this problem, if the event SDC 216 does not receive the begin bracket after commanding a recipe event recovery, it may continue to journal subsequent recipe events received and update its internal list if it has the next expected sequence identifier. It may not re-command another recipe event

recovery until it has received the begin bracket or until some time (such as 30 seconds) has passed from the first commanded recovery. It may be assumed that the recipe events that follow the recipe event begin bracket are always in sequence (if disruption does not occur).

[00101] Upon completion of a recipe execution, the control recipe may report a control recipe complete event with its batch identifier. The batch identifier may be stored in a unique identifier field in the notification structure. This event may have a special sub-type to indicate that it is the last event from the control recipe. Like other recipe events, it may also contain a sequence identifier, and the identifier may represent the next increasing sequence identifier that follows the last recipe event sent by the recipe execution. Additional details regarding the control recipe complete event are provided in Table 3.

Field	Value
NmCategory	NC_BATCH_EVENT
NmType	NM_TYPE_BATCH_EVENT
NmSubType	NM_SUBTYPE_CR_COMPLETE
NmEECode	Execution environment code
NmPriority	NM_PRIORITY_JOURNAL
NmUnikId	Batch identifier
NmSeqId	Sequence identifier of the last recipe event + 1
NmLoadTime	Load time of the execution environment
szAuxDesc	Description for this event, such as "Recipe completed"
NmValue	Batch identifier

Table 3

When the notification client 206 detects this event, it may set a flag in the notification queue packet associated with the control recipe complete event before posting it into the notification queue 208.

[00102] When this event reaches the event SDC 216, if the sequence identifier for the control recipe complete event matches the next sequence identifier expected by the

50

event SDC 216, it means that the event SDC 216 has successfully received and journaled all recipe events for the recipe. It may then call a hsc\_gda\_control\_recipe\_complete() GDA API function to  
5 signal the CDA server 204 that the control recipe can be deleted. The CDA server 204 may prevent the deletion of the control recipe until signaled by the event SDC 216. If the event SDC 216 detects a sequence identifier mismatch for the control recipe complete event, it means that not  
10 all recipe events for the recipe execution are journaled, and it may invoke a recipe event recovery to retrieve the lost events. There can be situations where a control recipe has completed its execution and the control recipe complete event is sent but is lost before it reaches the  
15 event SDC 216. As a result, the CDA server 204 may be stuck waiting for a signal from the event SDC 216. A mechanism to recover from this state can be provided, such as when the CDA server 204 or control execution environment re-announces the recipe complete event after some timeout.

20 **[00103]** The notification client 206 itself may also support various functions to enable guaranteed event delivery in the process control system. Among other things, this may include handling the recipe event recovery begin brackets and the control recipe complete events. As  
25 mentioned above, the notification client 206 may set a flag for the notification queue packets for these special events before putting them into the notification queue 208. The flags may be defined as follows.

```
prtque_def
30 // Recipe event recovery begin bracket event
#define PRQ_CR_RECOVERY_BRACKET_MASK 0x0004 // field mask
#define PRQ_CR_RECOVERY_BRACKET_ 0x0004
#define PRQ_CR_RECOVERY_BRACKET_BIT 3 // bit position

35 // Control recipe complete event
#define PRQ_CR_COMPLETE_EVT_MASK 0x0004 // field mask
```

51

```
#define PRQ_CR_COMPLETE_EVT 0x0004  
#define PRQ_CR_COMPLETE_EVT_BIT 4           // bit position
```

Two similar flags may be added to a definition file so that  
5 the notification manager 210, when it processes the  
notification packets for the two events from the  
notification queue 208, also sets the corresponding flags  
on the event file record structure before putting the  
events into the event list 214. Thus, the event SDC 216  
10 may know how to detect these events when they arrive and  
respond accordingly.

[00104] In situations when the CDA server 204 knows that  
a recipe event recovery is required, the CDA server 204 may  
not know the recipe event at which to start the recovery  
15 and may not know what events have been journaled. The CDA  
server 204 may obtain this from the event SDC 216, where  
the information is available. To achieve this, a callback  
function may be added to the notification client 206. The  
notification client 206 callback function, denoted  
20 nc\_get\_SeqID\_for\_EEs(), may provide the CDA server 204 an  
entry point for querying the sequence identifier of the  
last-journaled recipe event received from each controlled  
execution environment. This callback function can be  
passed to the CDA server 204 when the notification client  
25 206 subscribes to notification on server startup. As  
mentioned above, it may be passed as a parameter in the  
GNotificationSubscribe() function. The call function  
nc\_get\_SeqID\_for\_EEs() may receive a list of "recipe  
event-capable" execution environments and their load times  
30 as input parameters and may return a list of sequence  
identifiers that aid in recipe event recovery from the  
execution environments. It may be called by the CDA server  
204, for example, in the following situations: the  
notification client 206 subscribes to notification from the

CDA server 204, a controller restart/reconnect, and a controller cold reload. This kind of recipe event recovery may be considered as a CDA server-activated recovery. It may be slightly different from a recipe event recovery initiated by the event SDC 216 in that it is initiated in a different component due to different reasons. When the callback function is called, it may query the event SDC 216 for the list of sequence identifiers through a getSequenceIDs() method. It may then return this information to the CDA server 204. Upon receiving this information from the callback function, the CDA server 204 may request a recipe event recovery from the execution environments. The definition of the notification client 206 callback function is provided below.

```

15     int hsc_nc_get_SeqID_for_EEs(int NumOfEE, WORD* EEs,
        FILETIME* LoadTimes, UINT64** SequenceIDs)
        // Query event SDC to get the last sequence ID received from each
        // execution environment and return list of sequence IDs to CDA
        // server
20     // RETURN VALUES:
        // FP_GET_SEQID_ERROR if an error occurs when querying for the
        // sequence IDs
        // FP_GET_SEQID_OK if sequence IDs are obtain successfully
        // FP_GET_SEQID_CANCEL if Main Thread believes this thread should
25     // be dead

```

[00105] The GDA manager 220 may further support various functions to enable guaranteed event delivery in the process control system. For example, the GDA manager 220 may support API functions for accessing the CDA server 204. Two CDA API functions that may be used for event recovery could include: hsc\_cda\_recipe\_event\_recovery() and hsc\_cda\_control\_recipe\_complete(). These methods may call the GRecipeEventRecovery() and GControlRecipeComplete() CDA notification functions, respectively. Additional details about these CDA API functions are shown below.

```

        //-----
        //HSC_CDA_RECIPE_EVENT_RECOVERY - calls for a recipe event
        recovery

```

```

//-----
-
// SUMMARY:
// Call GRecipeEventRecovery
5 // If status is not OK then raise alarm
void hsc_cda_recipe_event_recovery(int2 EE, HSCTime LoadTime, int8
    SeqID);
//-----
-
10 //HSC_CDA_CONTROL_RECIPE_COMPLETE - signals that all recipe events
    // for the CR is journaled
//-----
-
// SUMMARY:
15 // Call GControlRecipeComplete
// If status is not OK then raise alarm
void hsc_cda_control_recipe_complete(int4 BatchID);

    [00106] Other methods that could be used or supported by
the GDA manager 220 may include RecipeEventRecover() and
20 ControlRecipeComplete(). Calling these two methods may
result in the corresponding hsc_cda_recipe_event_recovery()
and hsc_cda_control_recipe_complete() CDA API methods being
called. Below are more details about these two methods.

    HRESULT RecipeEventRecover (
25 //PARAMETERS
    [in] int2 EE, // Execution environment (EE) code
    [in] HSCTime LoadTime, // Load time for the EE
    [in] int8 SeqID, // Sequence IDs for recovery
    [out] GDAERR *pStatus); // Status returned for request
30 //REMARKS
    // Only CDA recipe event recoveries may be done via this
    // method. No GDA recoveries may be needed.
    //RETURN VALUES
    // S_OK - The method call was successful
35 // S_FALSE - The method call was partially successful, the
    // error is returned in pStatus.
    // failed HRESULT - The method call failed

    HRESULT ControlRecipeComplete (
40 //PARAMETERS
    [in] int4 BatchID, // Batch ID of the completed control recipe
    [out] GDAERR *pStatus); // Status returned for request
    //REMARKS
    // Only CDA control recipe complete signals may be done via
45 // this method. No GDA may support this method.
    //RETURN VALUES
    // S_OK - The method call was successful
    // S_FALSE - The method call was partially successful, the
    // error is returned in pStatus.
50 // failed HRESULT - The method call failed

```

```

HRESULT RecipeEventRecover (
    //PARAMETERS
    [in] int2 EE,           // EE code
    [in] HSCTime LoadTime, // Load time for the EE
    [in] int8 SeqID,       // Sequence IDs for recovery
    [out] GDAERR *pStatus); // Status returned for request
    //REMARKS
    // Clients call this method to initiate a recipe event
    // recovery from the server.
    //RETURN VALUES
    // S_OK - The method call was successful
    // failed HRESULT - The method call failed

HRESULT ControlRecipeComplete (
    //PARAMETERS
    [in] int4 BatchID,      // Batch ID
    [out] GDAERR *pStatus); // Status returned for request
    //REMARKS
    // Clients call this method to signal the CDA server that
    // all recipe events for the CR execution are journaled.
    //RETURN VALUES
    // S_OK - The method call was successful
    // failed HRESULT - The method call failed

```

**[00107]** Two additional GDA API functions could be used in the GDA manager: `hsc_gda_recipe_event_recovery()` and `hsc_gda_control_recipe_complete()`. These methods may call the `RecipeEventRecover()` and `ControlRecipeComplete()` GDA methods, respectively. Details about these two functions are provided below.

```

//-----
// HSC_GDA_RECIPE_EVENT_RECOVERY - calls for a recipe event
// recovery
//-----
void hsc_gda_recipe_event_recovery (
    int EE,           // EE code
    HSCTime LoadTime, // Load time for the EE
    int8 SeqID)       // Sequence ID for recovery

//-----
// HSC_GDA_CONTROL_RECIPE_COMPLETE - signals that all recipe
// events for the CR are journaled
//-----
void hsc_gda_control_recipe_complete (
    int4 BatchID) // Batch ID

```

**[00108]** The event SDC 216 may also support various functions to enable guaranteed event delivery in the process control system. To keep track of any lost recipe events, the event SDC 216 may need to maintain an in-memory



list containing the available "recipe event-capable" execution environments, the last sequence identifier received from each of these execution environments, and their load times. This list may provide fast lookup of the sequence identifier that the event SDC 216 is expecting from an execution environment. An example of the internal list in the event SDC 216 is shown in Table 4.

EE code	EE load time	Last received sequence ID
1	14/12/2004, 11:53:39.463	979574
14	24/01/2005, 19:35:23.535	0
16	03/11/2004, 06:12:48.032	35332623
:	:	:
:	:	:

Table 4

Note that the second record in the list above has a last received sequence identifier of zero. This means that the event SDC 216 has not journaled any recipe events from this execution environment before, or the execution environment has reloaded and the event SDC 216 is expecting the CDA server 204 to recover all recipe events from the execution environment. The list may be initialized when the server starts.

[00109] To keep track of all journaled recipe events from each "recipe event-capable" execution environment, a table may be used in an SQL or other database. The table may contain all execution environments that previously have had their recipe events journaled, the last sequence identifier journaled from each of these execution environments, and their execution environment load times. The layout of the table may be the same as the list above. After a time period elapses (such as every minute), the event SDC 216 may copy all entries in its in-memory list to this table. This may reduce or minimize the load placed on

the event SDC 216 to update the table. In terms of redundancy, this table may be synchronized to a backup server in the same manner as the event journal 218.

[00110] A getSequenceIDs() function of the event SDC 216  
5 may be called by the notification client 206 when the CDA server 204 tries to initiate a recipe event recovery. It may return the sequence identifiers of the last-journaled recipe events received from a requested list of execution environments. When this function is called, the event SDC  
10 216 may try to resolve this request using its internal list first. If an execution environment entry is found with the same requested execution environment load time, the event SDC 216 may return the last received sequence identifier from the list. If the execution environment load time is  
15 different, this may mean the controller has been reloaded, and the function may return a zero sequence identifier for the call to indicate that the event SDC 216 wants to recover all recipe events from the execution environment. For the latter case, the execution environment entry in the  
20 event SDC's internal list may be updated with the new load time for the execution environment and a zero sequence identifier. If the execution environment entry cannot be found in the list or the list has not been initialized, the event SDC 216 may make a query to the event journal 218 to  
25 obtain the largest sequence identifier journaled for the execution environment with the same requested execution environment load time. This value is then used in the appropriate entry in the event SDC's internal list. The query to the event journal 218 may be made using an  
30 ems\_sp\_GetGetSequenceID stored procedure. If the sequence identifier is found, the function may update the event SDC's internal list and return with that sequence identifier. Otherwise, a sequence identifier of zero may

be returned. Below are more details about the  
getSequenceIDs() function and the stored procedure.

```

// DESCRIPTION:
// Get the latest sequence ID for each of the EEs in the list
5 // PARAMETERS:
//   [in] short connectID,
//   [in] int NumOfEE,
//   [in] int* EEs,
//   [in] __int64* EELoadTimes,
10 // RETURN VALUES:
//   Standard com S_OK or E_FAIL.
//   [out] __int64** EESequenceIDs
[helpstring("Event SDC method getSequenceIDs")] HRESULT
getSequenceIDs(
15     [in] short connectID,
        [in] int NumOfEE,
        [in] int* EEs,
        [in] __int64* EELoadTimes,
        [out] __int64** EESequenceIDs);
20
ems_sp_GetGetSequenceID.sql
// NAME:   ems_sp_GetGetSequenceID
// DESCRIPTION:
// Retrieves the latest sequence ID for the EE given its
25 // load time
// PARAMETERS:
// @EE      (in)   The EE to get the latest sequence ID for
// @LoadTime (in)   The load time of the EE
// @SequenceID (out) The retrieved latest sequence ID
30

```

**[00111]** When the event SDC 216 processes recipe events, it may check for any gap in the sequence identifiers. If it detects a recipe event with a sequence identifier higher than the next expected sequence identifier, this may mean  
35 one or more events have been lost, and the event SDC 216 may command a recipe event recovery immediately. If the event SDC 216 receives the expected recipe event for the execution environment, the event SDC 216 may update its internal list with the sequence identifier of the event.  
40 If the event SDC 216 detects a recipe event with a sequence identifier lower than the last received sequence identifier in the list, it may do nothing to the list. In all three cases, the event SDC 216 may journal the recipe events into the event journal 218.

45 **[00112]** When the event SDC 216 receives a recipe event

recovery begin bracket, it may update its internal list with the sequence identifier stored in the event's value field if and only if the sequence identifier in the list is less than the received sequence identifier. This is because the sequence identifier in the list identifies the last recipe event journaled, and there may be no need to revert the sequence identifier to anything lower than that. The event SDC 216 may never receive a recovery begin bracket that has a different execution environment load time than contained in its internal list. This is because, whether the event SDC 216 commands or the CDA server 204 activates a recipe event recovery, the event SDC 216 may always update its internal list with the new execution environment load time first. When the recovery begin bracket reaches the event SDC 216, it may always contain the expected execution environment load time.

[00113] To support guaranteed event delivery, the following attributes may be added to event schema supported by the event journal 218: a batch identifier, a sequence identifier, and an execution environment load time. All of these attributes may be journaled into the event journal 218. Also, an additional parameter index attribute (such as a UINT32 value) may be added to the event schema, and the CDA notification field may be mapped to this attribute. This attribute in the event schema may also be journaled into the event journal 218.

[00114] As noted above, the CDA server 204 could use a notification structure that includes a sub-condition field, which could represent a string (such as LPSTR). This new notification field may be mapped into the existing sub-condition attribute in the event schema.

[00115] The following represents specific scenarios that might occur in the process control system 100. A control

recipe may start and generate recipe events that are journaled in the event journal 218. The controller, which is still running the control recipe, is then powered off (assuming no battery backup and an empty database. A  
5 complete reload of the controller is performed, and the control recipe is started again. All new recipe events from the execution environment need to be retrieved. Without the load time of the execution environment, the event SDC 216 may not know that the controller has been  
10 reloaded, and it may request an event recovery using the sequence identifier continued from the last recipe event received from the execution environment prior to the reload. This may cause some new recipe events generated from the new run of the control recipe to be missed during  
15 the recovery. To avoid this, the control execution environment could continue using the next sequence identifier even after a load, the batch identifier could be used instead of the load time, or the next sequence identifier could be persisted even after a reload of the  
20 controller. Also, the CDA server 204 may only have the list of execution environments and not their load times, and the event SDC 216 could return the sequence identifier associated with the latest load time for an execution environment. The CDA server 204 could also provide this  
25 load time to the event SDC 216 since the load time in the event SDC 216 may not be the execution environment's current load time and the event SDC 216 may need to know the current load time to identify the sequence identifier at which to start recovery. The CDA server 204 could  
30 obtain the load time from the execution environment.

[00116] In another example scenario, there could be situations where a control recipe has completed its execution and the control recipe complete event is sent but

is lost before it reaches the event SDC 216. As a result, the CDA server 204 may be stuck waiting for a signal from the event SDC 216 before the control recipe can be deleted. There could be a mechanism to recover from this state, such as having the CDA server 204 re-announce this event after some timeout.

[00117] In addition, the CDA server 204 may not hold on to any events (except in the case of console stations through a notification cache). As a result, there may be no way of even the CDA server 204 knowing that an event has been lost on its way to the notification client 206. The CDA server 204 could be enhanced so that every time it sends a recipe complete event to the notification client 206, it waits to get the `hsc_gda_control_recipe_complete()` call from the event SDC 216 within a specified time period (after which the CDA server 204 re-announces this event). Also, if a control execution environment does not receive the recipe complete indication from the event SDC 216 (through the CDA server 204), it may reissue the recipe complete event.

[00118] Although FIGURES 5 and 6 illustrate one specific example of a portion of a process control system supporting guaranteed batch event delivery, various changes may be made to FIGURES 5 and 6. For example, the process control system 500 could include any number of each of the components 502-518. Also, other suitable techniques could be used to implement or support guaranteed event delivery.

[00119] FIGURE 7 illustrates an example user interface 700 supporting guaranteed batch event delivery in a process control system. The embodiment of the user interface 700 shown in FIGURE 7 is for illustration only. Other embodiments of the user interface 700 may be used without departing from the scope of this disclosure.

[00120] In this example, the user interface 700 is associated with a control execution environment. The user interface 700 includes various tabs for accessing different information about and setting different operational aspects of the control execution environment. When the "main" tab is selected, the user interface 700 presents various information and selections to the user. One of the selections includes a drop-down menu 702, which can be used by the user to select the size of the event buffer 203 in the control execution environment. In this example, the drop-down menu 702 includes four different options for the buffer size: none, small, medium, and large. With the "none" option, a buffer size of zero is used, and guaranteed batch event delivery is disabled. In particular embodiments, the small, medium, and large buffers 203 can accommodate 120, 240, and 740 events, respectively. The user selects a proper buffer size based on, for example, the application in which the control execution environment is being used.

[00121] Although FIGURE 7 illustrates one example of a user interface 700 supporting guaranteed batch event delivery in a process control system, various changes may be made to FIGURE 7. For example, any other suitable user interface or other suitable technique could be used to set the size of the event buffer 203 in a control execution environment.

[00122] FIGURE 8 illustrates an example method 800 for guaranteed batch event delivery in a process control system. The embodiment of the method 800 shown in FIGURE 8 is for illustration only. Other embodiments of the method 800 could be used without departing from the scope of this disclosure.

[00123] A controller generates and buffers multiple

events at step 802. This may include, for example, a control execution environment generating various events associated with its operation, such as events associated with the execution of one or more recipe control modules 504 and sequential control modules 506.

[00124] The controller associates a unique identifier with each event at step 804. This may include, for example, the controller assigning sequential numerical values to a sequence of events generated by the controller. The sequence number for each event could be unique within the current execution of the control execution environment. The sequence number, a code identifying the control execution environment, and the control execution environment's load time may collectively represent an identifier that uniquely identifies an event in the entire process control system.

[00125] The controller communicates the events to a supervisory server at step 806, and the supervisory server receives at least some of the events and determines if any events are missing at step 808. This could include, for example, the control execution environment providing the events to the notification client 206 for storage in the notification queue 208. This could also include the notification manager 210 retrieving the events from the notification queue 208. This could further include the notification manager 210 storing the events in the event list 214. In addition, this could include the event SDC 216 determining if any events are missing using the unique identifiers of the received events.

[00126] If no events are missing at step 810, the events are journaled at step 812. This could include, for example, the event SDC 216 storing the events in the journal 218. At this point, the process returns to step



802 where the controller generates more events. Otherwise, one or more events are missing, and the supervisory server communicates a request for the missing events to the controller at step 814. This may include, for example, the  
5 supervisory server sending a request for an event recovery. The request could be sent to the CDA server 204 through the notification client 206 or the GDA manager 220.

[00127] The controller receives the recovery request and attempts to locate the missing events at step 816. This  
10 may include, for example, the CDA server 204 causing the control execution environment to examine the events in its buffer 203, looking for events to be retransmitted. If all missing events are found at step 818, the controller transmits a begin bracket and the requested events,  
15 including the missing events, at step 820. Otherwise, one or more missing events cannot be found, such as when one or more events have been overwritten in the event buffer 203. In that case, the controller transmits a begin bracket and any requested events that could be found (if any) and sets  
20 an alarm at step 822. If none of the missing events could be found in the buffer 203, the transmission of the begin bracket and the requested events could be omitted in step 820. The alarm set in step 822 could, for example, inform  
25 a user of the missing events, allowing the user to take any suitable action. The recovered events are journaled at step 812, and the process returns to step 802 where the controller generates more events.

[00128] Although FIGURE 8 illustrates one example of a method 800 for guaranteed batch event delivery in a process  
30 control system, various changes may be made to FIGURE 8. For example, while shown as a series of steps, various steps in FIGURE 8 could overlap, occur in parallel, or occur in a different order. As a particular example, the

controller could continuously generate, buffer, and communicate events to the supervisory server, and the controller could intermittently retransmit lost or missing events to the supervisory server as needed.

5       **[00129]** In some embodiments, various functions are implemented or supported by a computer program that is formed from computer readable program code and that is embodied in a computer readable medium. The phrase "computer readable program code" includes any type of  
10 computer code, including source code, object code, and executable code. The phrase "computer readable medium" includes any type of medium capable of being accessed by a computer, such as read only memory (ROM), random access memory (RAM), a hard disk drive, a compact disc (CD), a  
15 digital video disc (DVD), or any other type of memory.

**[00130]** It may be advantageous to set forth definitions of certain words and phrases used throughout this patent document. The term "couple" and its derivatives refer to any direct or indirect communication between two or more  
20 elements, whether or not those elements are in physical contact with one another. The terms "application" and "program" refer to one or more computer programs, software components, sets of instructions, procedures, functions, objects, classes, instances, related data, or a portion  
25 thereof adapted for implementation in a suitable computer code (including source code, object code, or executable code). The terms "transmit," "receive," and "communicate," as well as derivatives thereof, encompass both direct and indirect communication. The terms "include" and  
30 "comprise," as well as derivatives thereof, mean inclusion without limitation. The term "or" is inclusive, meaning and/or. The phrases "associated with" and "associated therewith," as well as derivatives thereof, may mean to

include, be included within, interconnect with, contain, be  
contained within, connect to or with, couple to or with, be  
communicable with, cooperate with, interleave, juxtapose,  
be proximate to, be bound to or with, have, have a property  
5 of, or the like. The term "controller" means any device,  
system, or part thereof that controls at least one  
operation. A controller may be implemented in hardware,  
firmware, software, or some combination of at least two of  
the same. The functionality associated with any particular  
10 controller may be centralized or distributed, whether  
locally or remotely.

[00131] While this disclosure has described certain  
embodiments and generally associated methods, alterations  
and permutations of these embodiments and methods will be  
15 apparent to those skilled in the art. Accordingly, the  
above description of example embodiments does not define or  
constrain this disclosure. Other changes, substitutions,  
and alterations are also possible without departing from  
the spirit and scope of this disclosure, as defined by the  
20 following claims.

## WHAT IS CLAIMED IS:

1. A method comprising:  
generating multiple events associated with operation  
of a controller in a process control system;  
5 storing the events in a buffer;  
communicating the events to a supervisory device  
associated with the controller;  
receiving a request to recover one or more of the  
events;  
10 recovering at least one of the events from the buffer;  
and  
communicating the at least one recovered event to the  
supervisory device.
- 15 2. The method of Claim 1, wherein:  
each event is associated with a unique identifier; and  
the request identifies the unique identifier  
associated with one of the events.
- 20 3. The method of Claim 2, wherein:  
the unique identifier identified in the request  
comprises a numerical identifier; and  
recovering the at least one event from the buffer  
comprises recovering any event associated with a numerical  
25 identifier that is greater than the numerical identifier  
identified in the request.
4. The method of Claim 1, further comprising:  
generating an alarm in response to recovering less  
30 than all desired events from the buffer.

5. The method of Claim 1, wherein the events are associated with execution of at least one of:

a recipe control module that defines production requirements for a batch of a product; and

5 a sequential control module that provides access to a control module that controls a process element in the process control system.

6. The method of Claim 5, further comprising:

10 preventing deletion of the recipe control module until all events generated by the recipe control module have been journaled by the supervisory device.

7. The method of Claim 1, further comprising:

15 creating the buffer, the buffer having a user-specified size.

8. An apparatus comprising:

20 at least one memory configured to store a unique identifier associated with a first event received from a controller in a process control system; and

at least one processor configured to:

receive a second event from the controller;

25 determine whether any events between the first event and the second event have not been received from the controller using the unique identifier; and

initiate an event recovery if at least one event between the first event and the second event has not been received from the controller.

9. The apparatus of Claim 8, wherein:

the unique identifier associated with the first event comprises a first numerical identifier;

the first event comprises a most recently received  
5 event from the controller; and

the at least one processor is configured to determine whether any events between the first event and the second event have not been received from the controller by determining if a second numerical identifier associated  
10 with the second event is one greater than the first numerical identifier.

10. The apparatus of Claim 8, wherein the at least one processor is further configured to receive at least one  
15 recovered event from the controller during the event recovery.

11. The apparatus of Claim 10, wherein:

the unique identifier associated with the first event  
20 comprises a first numerical identifier;

the at least one processor is configured to initiate the event recovery by identifying the first numerical identifier; and

each recovered event is associated with a numerical  
25 identifier that is greater than the first numerical identifier.

12. The apparatus of Claim 8, wherein the apparatus comprises a supervisory device associated with the  
30 controller in the process control system, the supervisory device configured to journal the events in a database.

13. The apparatus of Claim 8, wherein:

at least some of the events are associated with execution in the controller of a recipe control module that defines production requirements for a batch of a product;

5 and

the at least one processor is configured to provide the controller with a signal indicating that all events generated by the recipe control module have been received from the controller, the controller configured to prevent  
10 deletion of the recipe control module until the signal has been received.

14. A computer program embodied on a computer readable medium, the computer program comprising:

15 computer readable program code for generating multiple events associated with operation of a controller in a process control system;

computer readable program code for storing the events in a buffer;

20 computer readable program code for communicating the events to a supervisory device associated with the controller;

computer readable program code for receiving a request to recover one or more events;

25 computer readable program code for recovering at least one of the events from the buffer; and

computer readable program code for communicating the at least one recovered event to the supervisory device.

70

15. The computer program of Claim 14, wherein:  
each event is associated with a unique identifier; and  
the request identifies the unique identifier  
associated with one of the events.

5

16. The computer program of Claim 15, wherein:  
the unique identifier identified in the request  
comprises a numerical identifier; and

the computer readable program code for recovering the  
10 at least one event from the buffer comprises computer  
readable program code for recovering any event associated  
with a numerical identifier that is greater than the  
numerical identifier identified in the request.

15 17. The computer program of Claim 14, further  
comprising:

computer readable program code for generating an alarm  
in response to recovering less than all desired events from  
the buffer.

20

18. The computer program of Claim 14, further  
comprising:

computer readable program code for creating the  
buffer, the buffer having a user-specified size.

25



19. The computer program of Claim 14, wherein:

at least some of the events are associated with execution in the controller of a recipe control module that defines production requirements for a batch of a product;

5 and

further comprising computer readable program code for preventing deletion of the recipe control module until all events generated by the recipe control module have been journaled by the supervisory device.

10

20. A system comprising:

a controller configured to control one or more process elements in a process control system, the controller also configured to generate and buffer multiple events associated with operation of the controller, each event associated with a unique identifier; and

15

a supervisory device configured to receive at least some of the events from the controller and to determine if any of the events have not been received from the controller based on the unique identifiers, the supervisory device also configured to initiate an event recovery by the controller if at least one of the events has not been received from the controller;

20

wherein the controller is further configured to recover at least one event and to communicate the at least one recovered event to the supervisory device during the event recovery.

25

21. The system of Claim 20, wherein:

the unique identifiers comprise numerical identifiers;  
and

the supervisory device is configured to determine  
5 whether any of the events have not been received from the  
controller by determining if a gap exists in the numerical  
identifiers associated with the events received from the  
controller.

10 22. The system of Claim 21, wherein:

the supervisory device is configured to initiate the  
event recovery by identifying the numerical identifier  
associated with one of the events; and

the controller is configured to recover any event  
15 having a numerical identifier greater than the numerical  
identifier identified by the supervisory device.

23. The system of Claim 20, wherein the controller is  
further configured to generate an alarm in response to  
20 recovering less than all desired events.

1/11

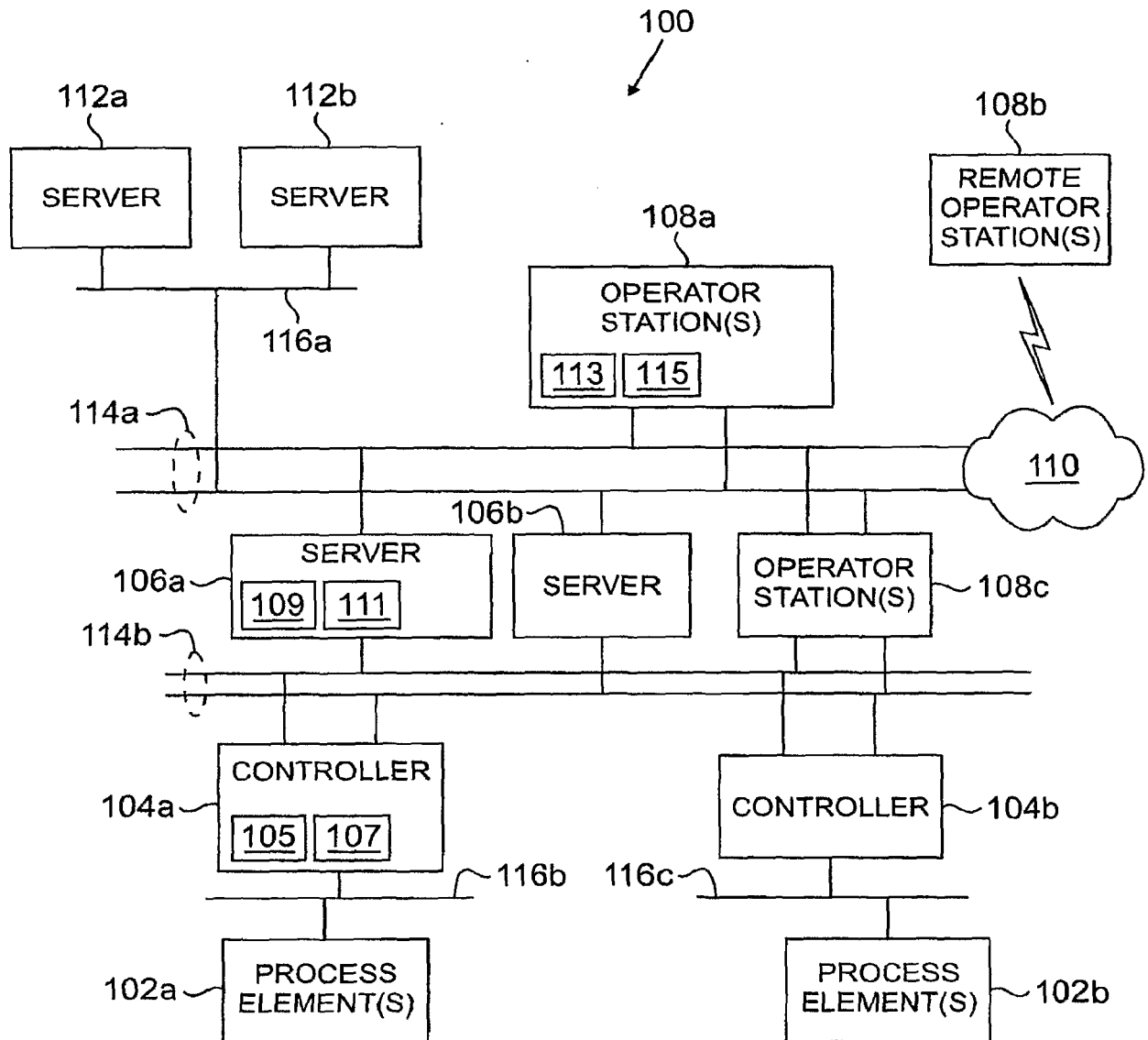
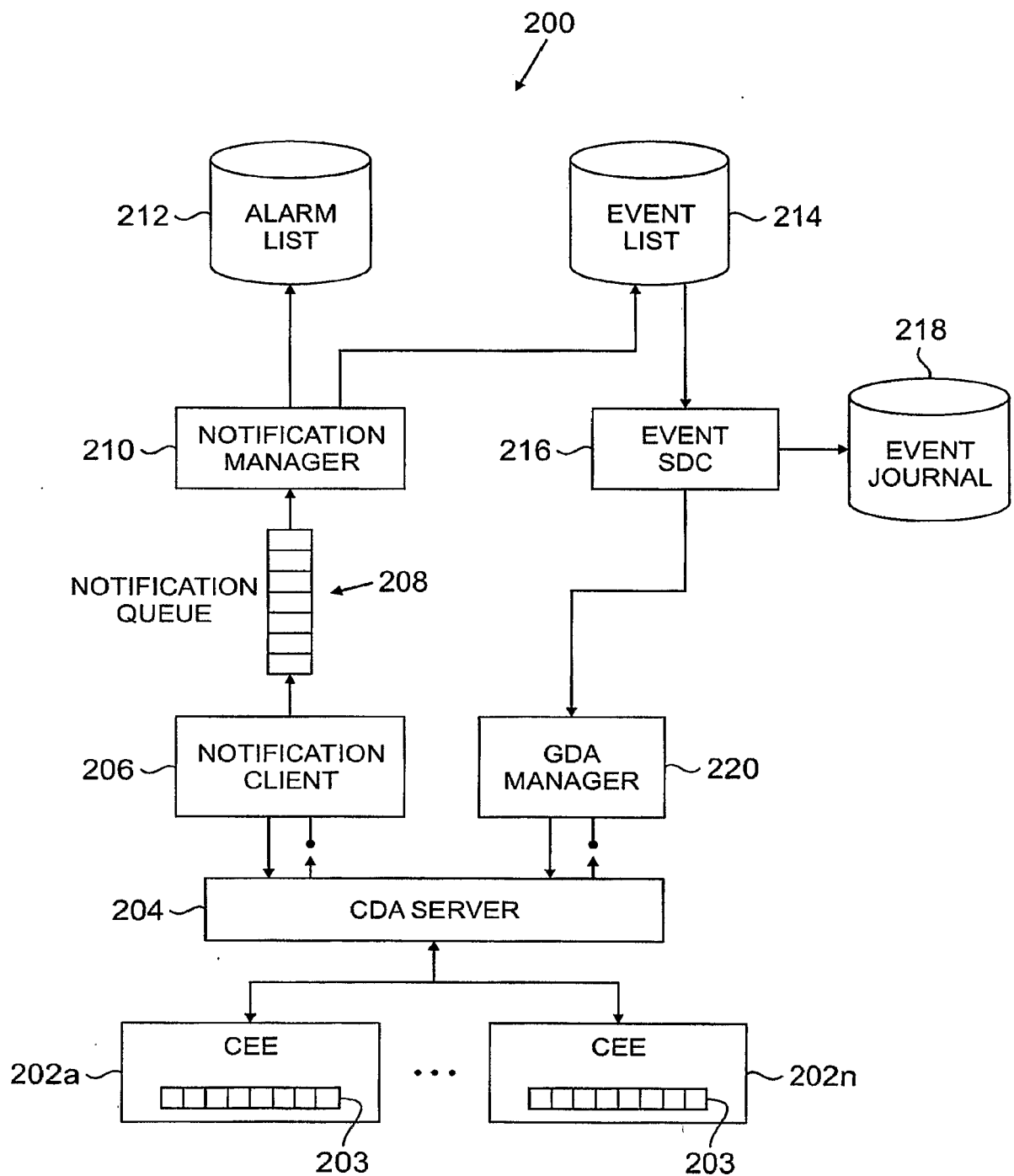


FIG. 1

2/11

**FIG. 2**

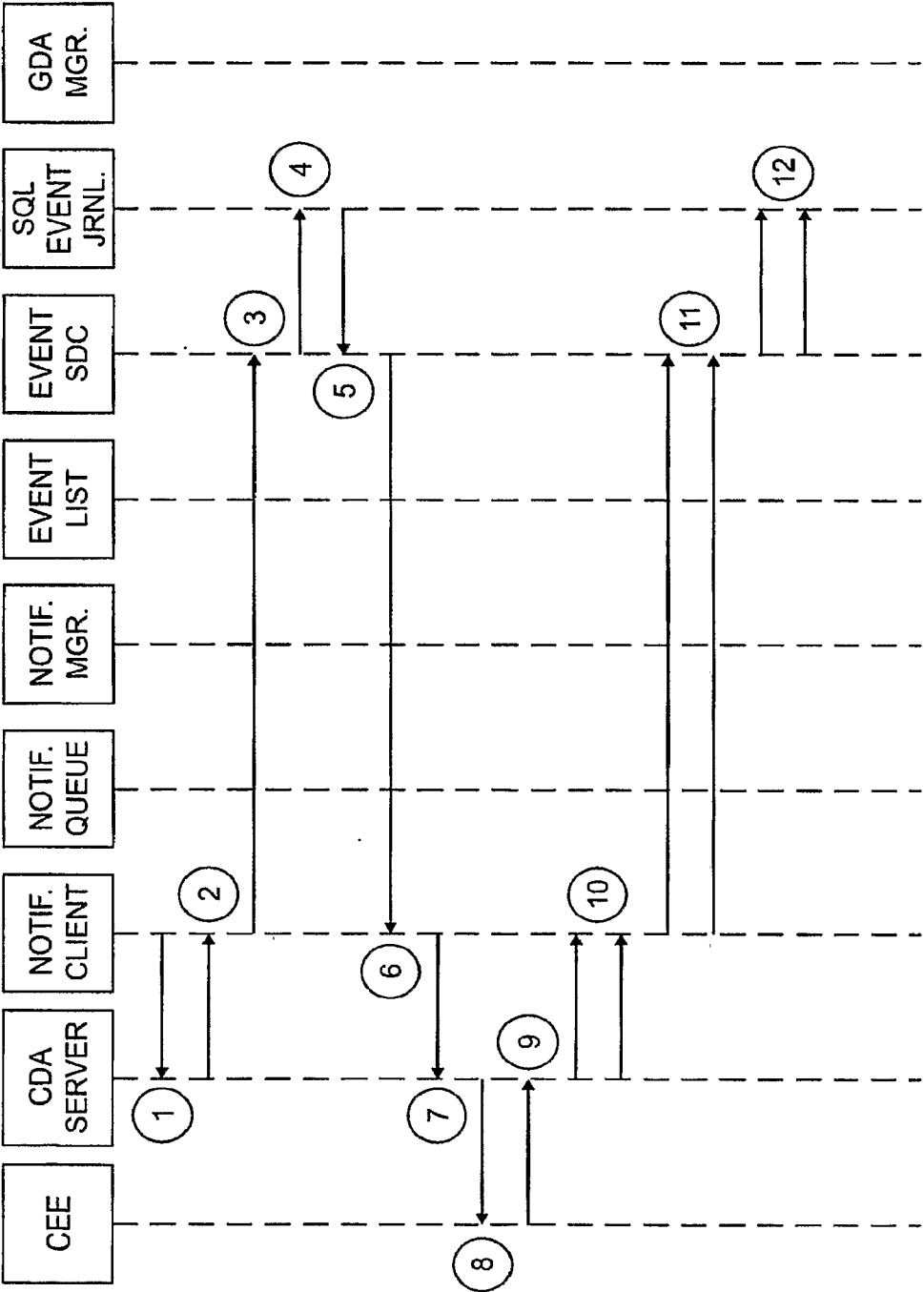


FIG. 3A

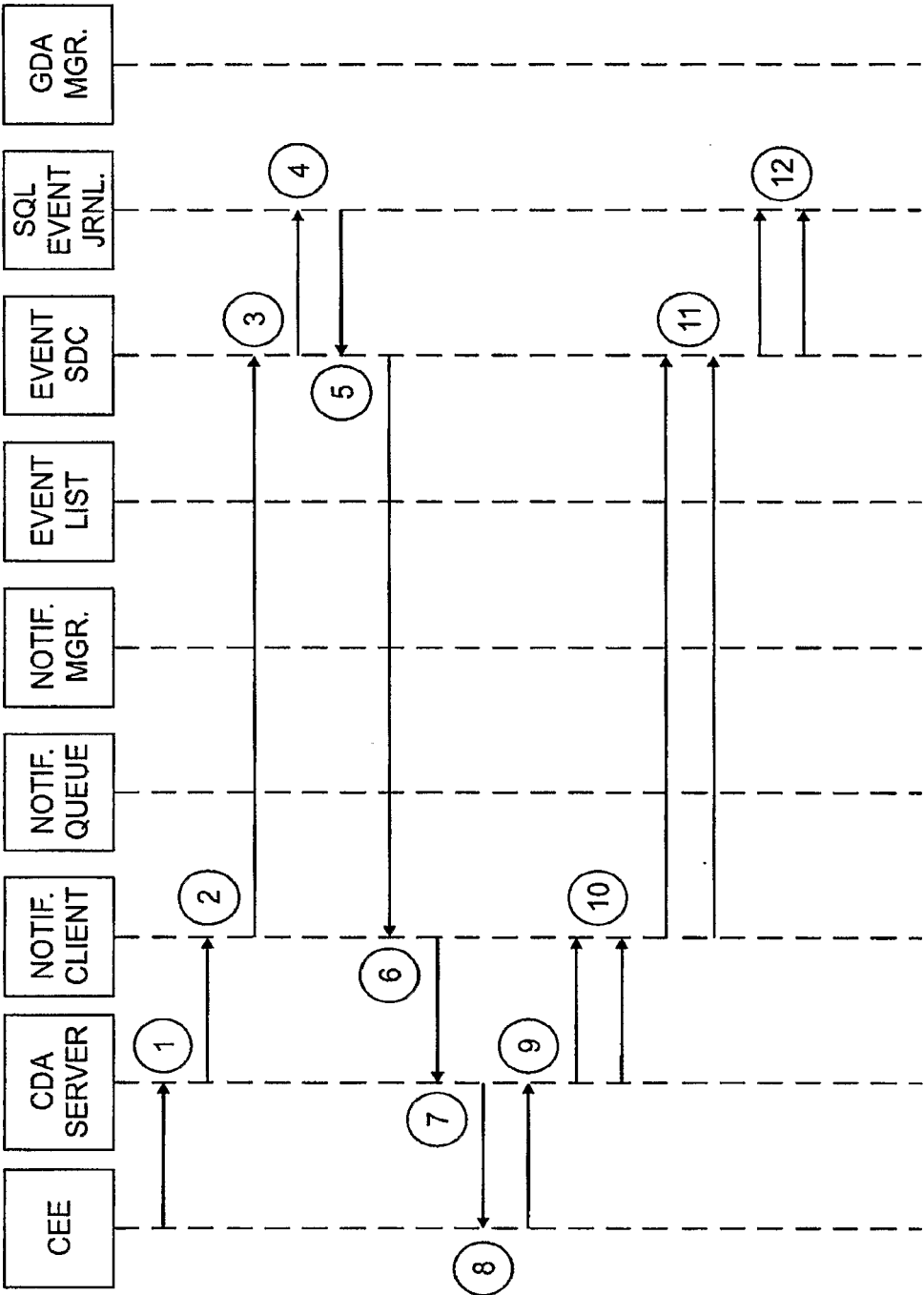


FIG. 3B



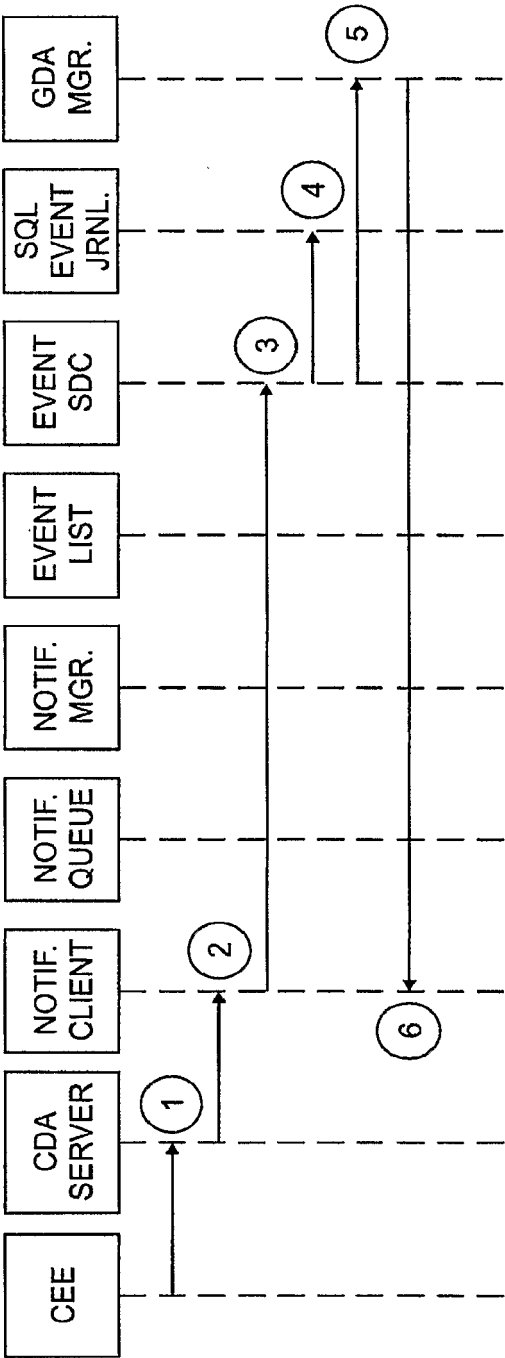


FIG. 3D



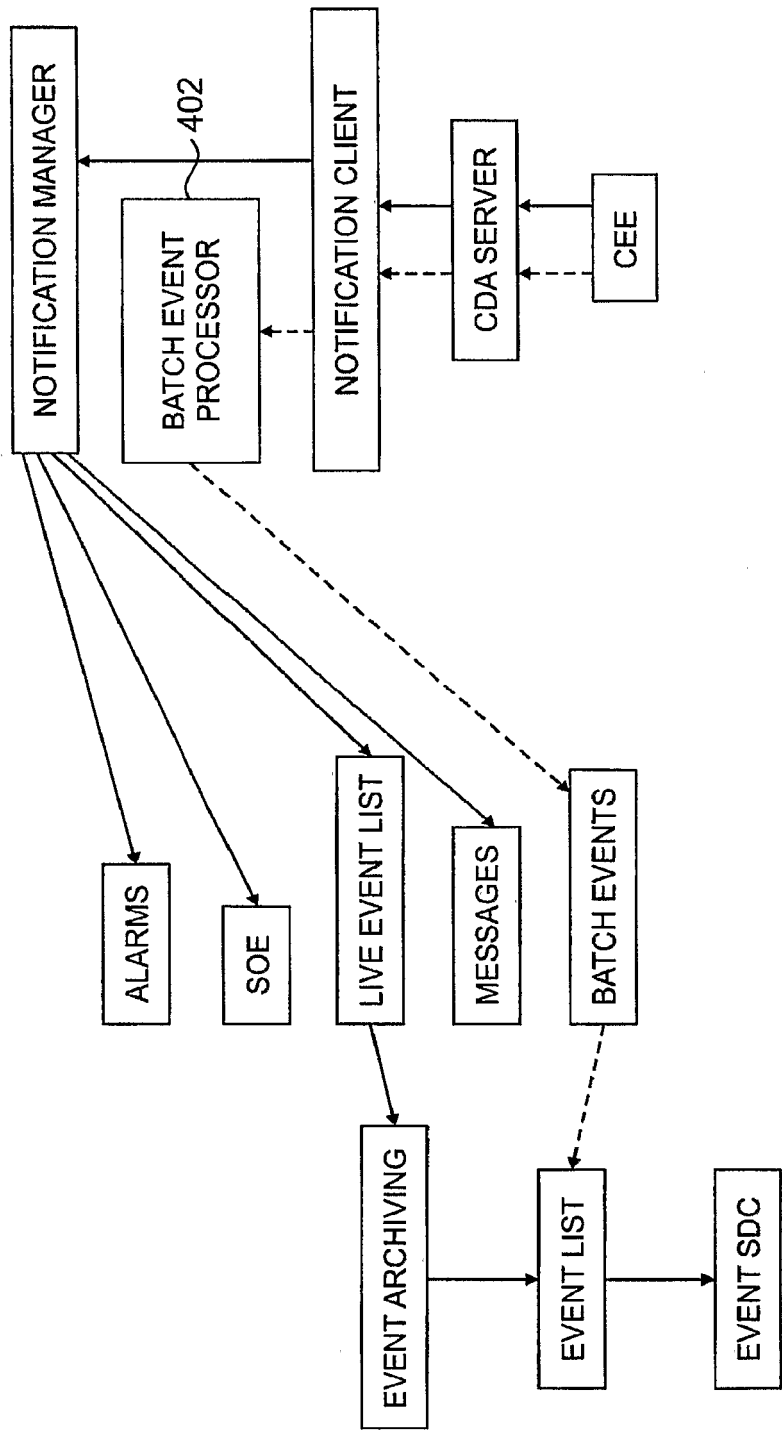


FIG. 4

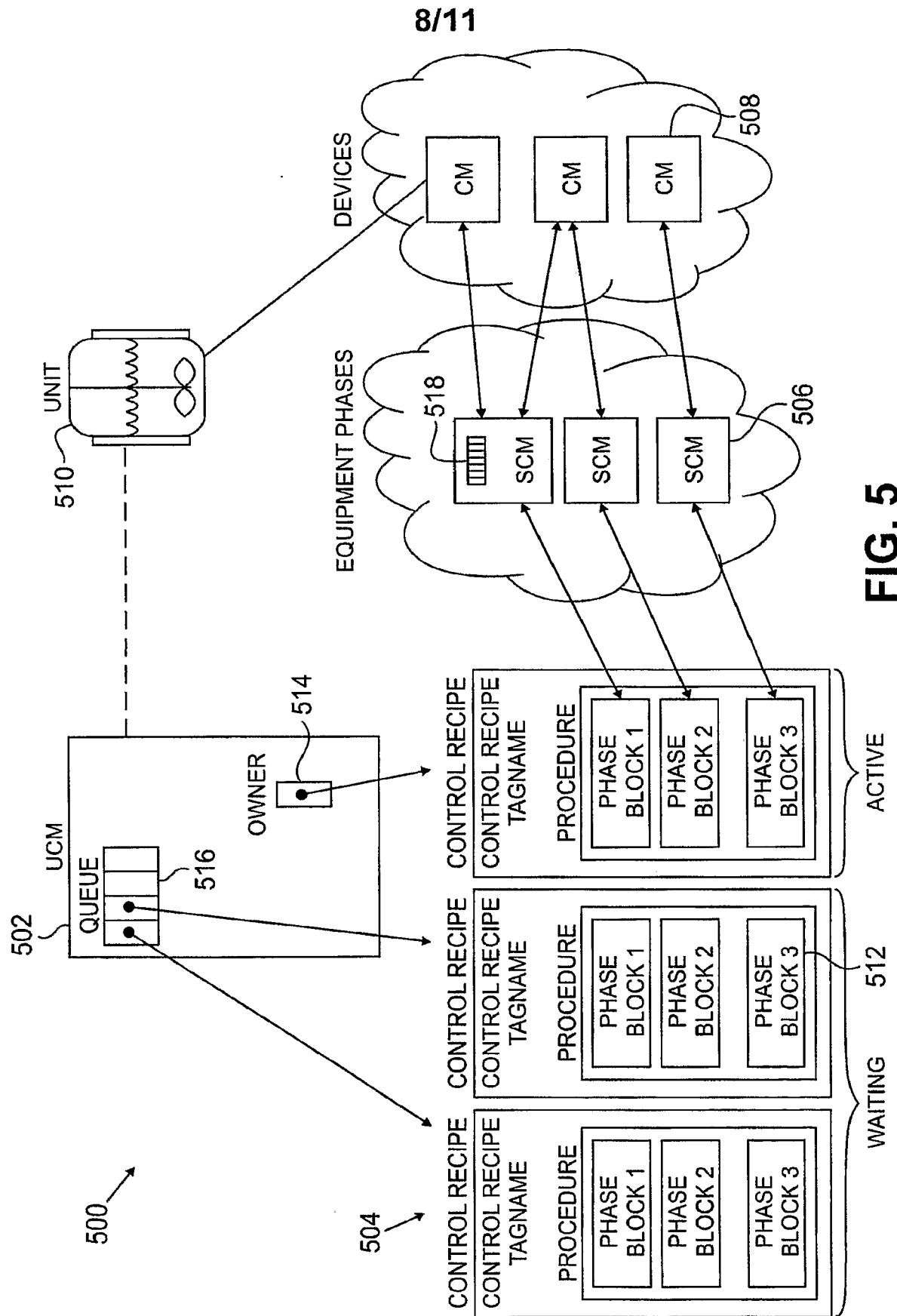


FIG. 5

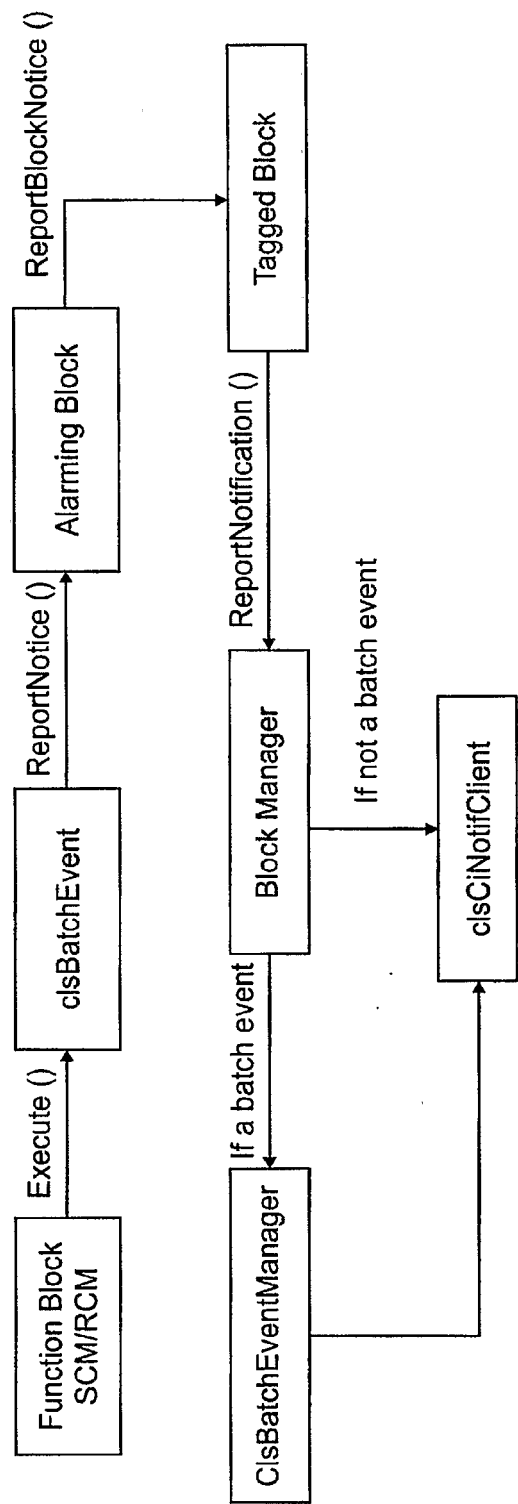


FIG. 6

**Control Builder - Project - Assignment**

**Project - Assignment**

SYSTEM: CEEC300 Block, CEEC300\_1455 - Parameters [Project]

Display Communications | Block Types Info | Server History | Server Displays | Control Confirmation | Identification  
Main | Peer Configuration | Statistics | CPU Loading | CPU Overruns | Memory | Peer Communications

Tag Name	CEE300_1455
Item Name	
Base Execution Period	50mS
Command/State	IDLE
CEE Command	IDLE
CEE State	Operator
User Lock for CEE Run	Supervisor
User Lock for CEE Idle	
<input checked="" type="checkbox"/> Program Access may command CEE from Idle to Run	
<input checked="" type="checkbox"/> Program Access may command CEE from Run to Idle	

Alarm Info

In-Alarm Flag OFF

☒ Alarming Enabled

☒ Enable Memory Limit Exceeded Alarm

Batch Event Settings

Batch Events Memory

Small None Small Medium Large

Powerup Restart Settings

CEE State IDLE

Warm Timeout

Time Info

Time Zone 0

☐ Daylight Savings Time

Year Format YYYY

Weekday Format 1Sunday

☐ Show Parameter Names

OK Cancel Help

**FIG. 7**

✓ 700

11/11

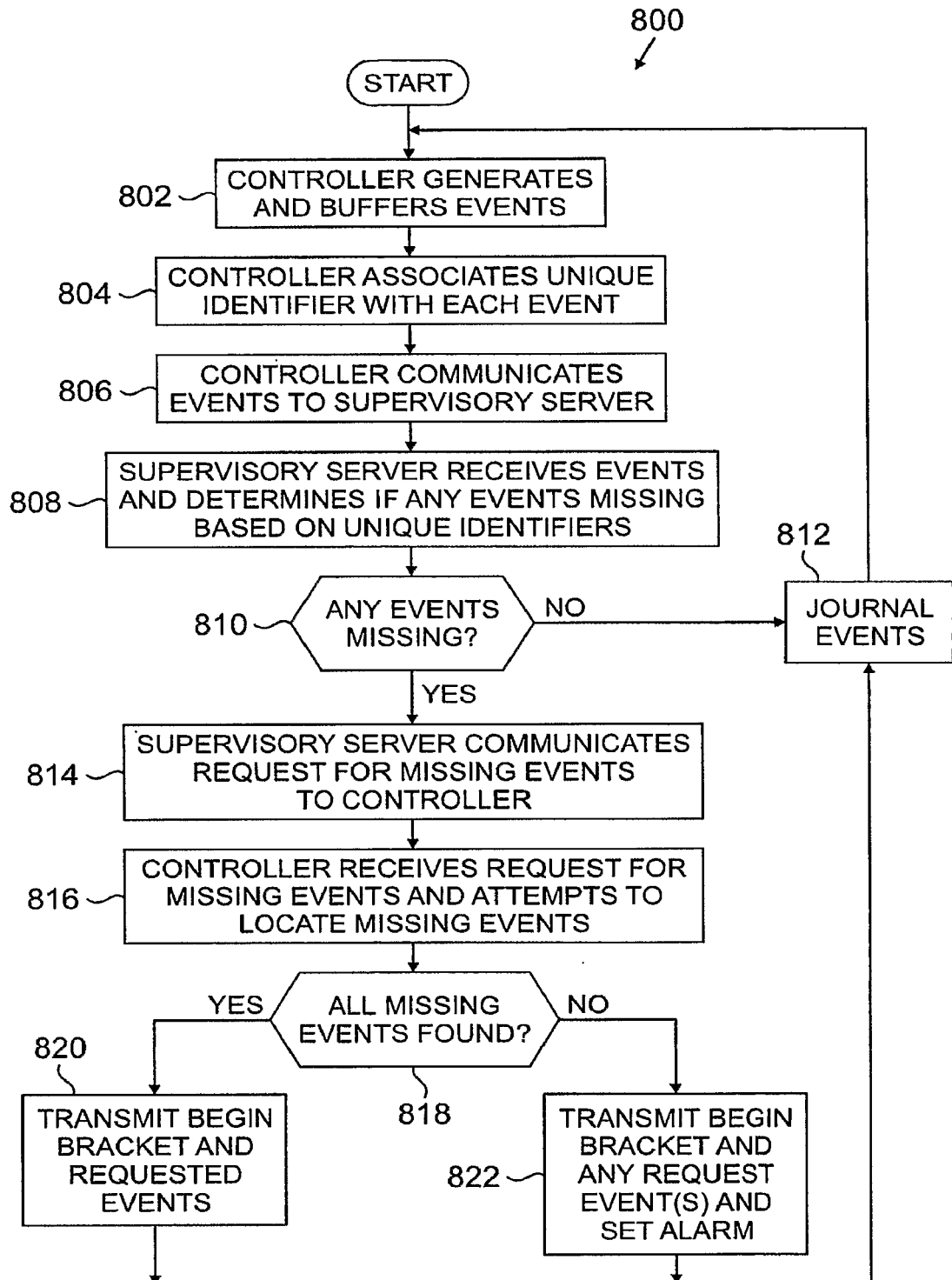


FIG. 8