

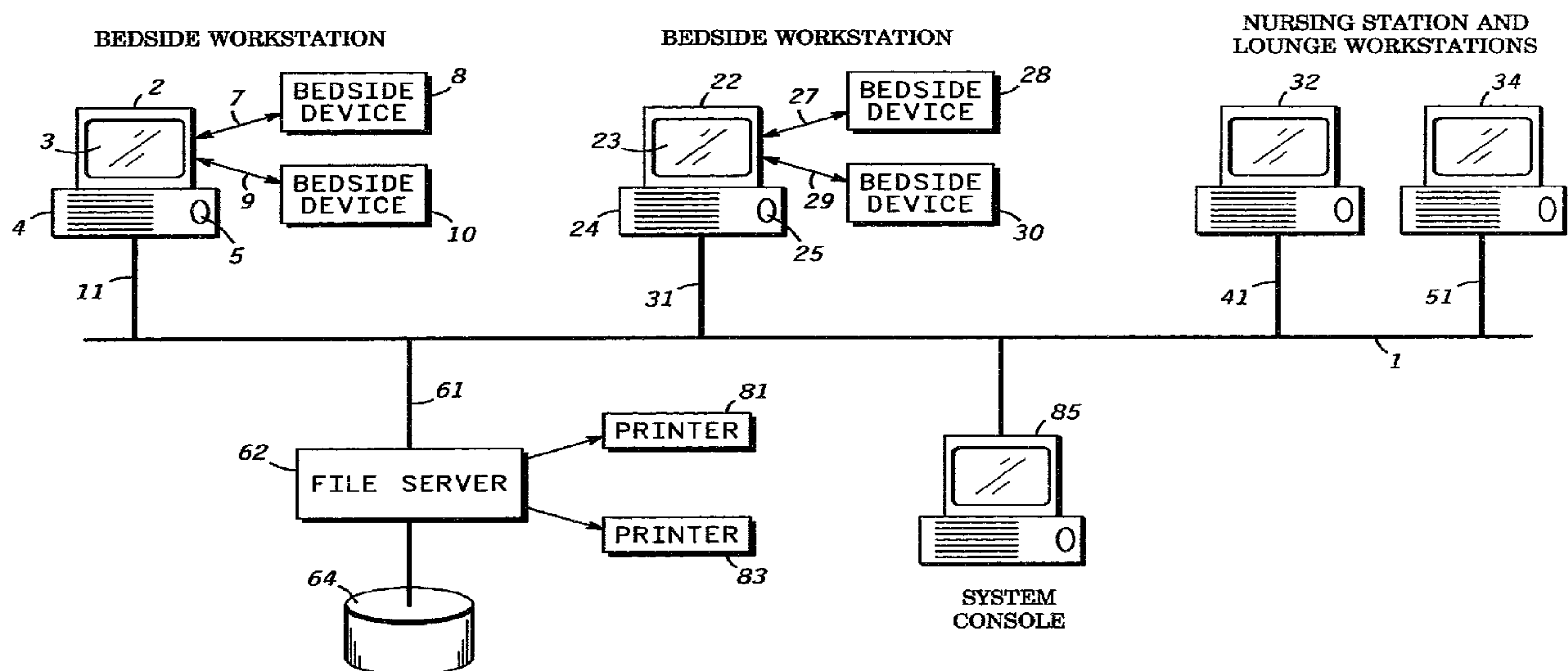


(11) (21) (C) **2,027,751**
(22) 1990/10/16
(43) 1991/05/31
(45) 2001/01/23

(72) Norden-Paul, Ronald Evan, US
(72) Person, Stanley Carl, US
(72) Wissinger, Robert A., US
(73) Emtek Health Care Systems, Inc., US

(51) Int.Cl.⁵ G06F 9/40
(30) 1989/11/30 (443,613) US
(30) 1989/11/30 (443,614) US
(30) 1989/11/30 (443,617) US
(30) 1989/11/30 (443,622) US
(30) 1989/11/30 (443,623) US
(30) 1989/11/30 (443,628) US
(30) 1989/11/30 (443,629) US
(30) 1989/11/30 (443,630) US

(54) **ASSOCIATION DE FONCTIONS DE REGLAGE IMPLICITE,
D'EDITION, DE CALCUL ET DE LISTAGE ET DE
COMBINAISONS DE CES FONCTIONS, AVEC DES
ATTRIBUTS ET DES TYPES D'ATTRIBUT**
(54) **TABLE MODIFIABLE EDIT FUNCTIONS WITH
ORDER-EFFECTIVE EDIT RULES**



(57) In a data processing system in which data is entered into one or more fields of a form and validated, a method for performing default, edit, calculation, and list procedures, and combinations thereof, according to which default, edit, calculation, and list procedures are associated with input attributes via a table. The procedures may be executed in a predetermined, yet modifiable order. Entry of certain data types may result in changes to multiple fields in a form. Upon entry of such data, a cross-field edit is performed to check for consistency between such data and other data in the database associated with the form.



TABLE MODIFIABLE EDIT FUNCTIONS WITH ORDER-EFFECTIVE EDIT RULES

5

ABSTRACT OF THE DISCLOSURE

10

In a data processing system in which data is entered into one or more fields of a form and validated, a method for performing default, edit, calculation, and list procedures, and combinations thereof, according to which default, edit, calculation, and list procedures are associated with input attributes via a table. The procedures may be executed in a predetermined, yet modifiable order. Entry of certain data types may result in changes to multiple fields in a form.

15

Upon entry of such data, a cross-field edit is performed to check for consistency between such data and other data in the database associated with the form.

TABLE MODIFIABLE EDIT FUNCTIONS WITH ORDER-EFFECTIVE EDIT RULES

5

Related Invention

10 The present invention is related to the following inventions, also assigned to the assignee of the present invention: Method for Generating Patient-Specific Flowsheets by Adding/Deleting Parameters, U.S. Pat. No. 4,878,175.

Technical Field

15

20 This invention relates generally to data processing systems and processes, and, in particular, to a data processing system in which various default, edit, calculation, and list procedures may be enabled as data is entered into one or more fields of a form.

Problem Addressed by the Invention

25 The present invention has utility in data processing systems in which various routine procedures are invoked in response to user input. The user input may take the form of one or more "attributes", i.e. the smallest piece of useful data in a data object (i.e. data field in the database). Refer to Glossary appended hereto as Appendix A.

30 Traditionally, libraries are built for routine procedures (such as defaults, edits, calculations, and lists, which hereinafter may be referred to DECL's). Each function is written to process one particular attribute. All applications must call the appropriate functions for the attributes they use. When a library function is changed, added, or deleted, all applications that use the function (i.e. the attribute the

35

function is associated with) must be correctly modified, recompiled, and retested.

5 Using the present invention, routine functions are associated with attributes and are automatically invoked by a separate process. The application never deals with them. Also routine functions can be associated with more than one attribute or an attribute type, so that they can be reused.

10 As a result, responsibility for routine tasks is removed from the applications. There is a corresponding reduction in errors, maintenance time, implementation time, and design time. There is also a reduction in complexity, because DECL's are reuseable (can be used with more than one attribute), so not as many are needed. A library of nodes, or program parts, is built up as applications are constructed. In
15 addition, the code is more readable and understandable.

 By reducing the number of DECL's procedures in a system, maintainability is enhanced, and the implementation of applications is speeded up. Instead of writing new DECL's procedures, the applications designer can
20 reuse existing ones. This increases programmer productivity, since the applications programmer is insulated from details he or she doesn't need to know.

 Further, the present invention enhances the ability to customize and configure systems for different customer
25 installations.

 It is known to utilize UNIX shell script in the form of a collection of separate and independent processes combined (e.g., filters and pipes) to achieve a particular purpose. However, UNIX shell script is significantly different from the
30 methods disclosed by the present invention. UNIX is oriented to a line-by-line style of interface where the control thread is controlled by the program rather than by a user with a mouse-driven interface. Also, UNIX is not field-oriented but rather is oriented to a stream of data.

35 It is also known to associate routine procedures with attributes in a form definition as part of related code.

However, such routine procedures are tightly-coupled to the attributes. They are not reuseable in the sense described herein regarding the present invention.

It is also known to associate routine procedures with attributes in the form of a table in a standard relational database such as SQL. Again, however, such routine procedures are tightly-coupled to the attributes and are not reuseable.

Brief Summary of Invention

Routine procedures (such as defaults, edits, calculations, and lists) are associated with attributes via a table and are executed by an applications manager process in response to events. All of the routine procedures are transparent to the application. The application doesn't have to know they exist.

Also routine procedures can be written so that they can process more than one attribute or any attribute of a particular type, thus reducing the number of procedures required.

The present invention can be used in any system that performs processing automatically on data entry, i.e. wherein the processing is "data driven".

Accordingly, it is an object of the present invention to provide a software architecture which removes responsibility for routine procedures from the applications.

It is another object of the present invention to provide a software architecture in which routine edit procedures are associated with attributes via a table and are executed by an applications manager process in response to events.

It is another object of the present invention to provide a software architecture in which routine procedures are associated with attribute types via a table and are executed by an applications manager process in response to events.

It is another object of the present invention to provide a software architecture in which edit procedures are

associated with attributes via a table and are executed by an applications manager process in a predetermined priority in response to events.

5 It is a further object of the present invention to provide a software architecture in which routine default procedures can be associated with attribute types.

10 It is another object of the present invention to provide a software architecture in which routine calculation procedures are associated with attributes via a table and are executed by an applications manager process in response to events.

15 It is another object of the present invention to provide a software architecture in which routine list procedures are associated with attributes via a table and are executed by an applications manager process in response to events.

 It is a further object of the present invention to provide a software architecture in which routine list procedures can be associated with attribute types.

20 It is another object of the present invention to provide a software architecture in which routine cross-field edit procedures are associated with attributes via a table and are executed by an applications manager process in response to events.

25 It is also an object of the present invention to provide a software architecture in which routine default, edit, calculation, and list procedures are associated with attributes via a table and are executed by an applications manager process in a predetermined priority in response to events.

30 These and other objects are achieved in accordance with a preferred embodiment of the invention by providing in a data processing system in which data is input and validated by one or more edit procedures, a method for performing the edit procedures, wherein the method
35 comprises the steps of categorizing the input data into one of a plurality of attributes; providing a table of edit procedures

for the plurality of attributes; when a given attribute is input, entering a routine to check the table to determine whether it contains any edit procedures for the attribute; if so, applying the edit procedures to the attribute; and if not, exiting the routine.

Brief Description of the Drawings

The invention is pointed out with particularity in the appended claims. However, other features of the invention will become more apparent and the invention will be best understood by referring to the following detailed description in conjunction with the accompanying drawings in which:

FIG. 1 shows a block diagram illustrating, in general, the patient care process in a typical hospital environment.

FIG. 2 shows a block diagram illustrating a preferred embodiment of a data processing system incorporating the method of the present invention.

FIG. 3 shows a typical Flowsheet information screen, illustrating both measured and calculated patient parameters in the automated clinical records system shown in FIGS. 1 and 2.

FIG. 4 shows a block diagram illustrating the relationship of the forms manager to the application program and database.

FIG. 5 shows a block diagram of the application software architecture of the present invention.

FIGS. 6A and 6B illustrate two function graphs.

FIG. 7 illustrates the structure of a dialog node

FIG. 8 illustrates the structure of a default procedure.

FIG. 9 illustrates the structure of an edit procedure.

FIG. 10 illustrates the structure of a calculation procedure.

FIG. 11 illustrates the structure of a list procedure.

FIG. 12 shows the "Graph Sequencing" function of graph management.

FIG. 13 shows the "Manage Input Dialog" function of graph management.

FIG. 14 shows a "Dialog Node".

FIG. 15 shows a "Default Procedures" node.

5 FIG. 16 shows a "List Procedures" node.

FIG. 17 shows an "Edit Procedures" node.

10 FIGS. 18A-18D show a flow diagram corresponding to the pseudo-code of Appendix B, illustrating how DECL's functions are associated with attributes according to a preferred embodiment of the invention.

Background Description of the Data Processing System

Patient Care Process

15

FIGS. 1-4 provide background description of a system in which the present invention has been implemented and provides utility. Such system is further described in the Related Inventions cited above. However, it should be understood that the present invention has broad utility in any data processing system which must be capable of a high degree of customization and is not intended to be limited to implementations such as those relating to the system shown in FIGS. 1-4.

20

FIG. 1 shows a block diagram illustrating in general the patient care process in a typical hospital environment.

25

Physician 301 is responsible for the management and care of the patient 315 throughout the hospital stay. On admitting a patient to the hospital a physician will assess the patient and arrive at a diagnosis. Based on the diagnosis the physician will write a plan of care in his progress notes and generate a list of physician orders 306 to be executed by nursing and other hospital personnel 303. The physician's orders 306 are the source for the Kardex 310, Task List 312, Requisitions 314, and other associated forms that assist in the overall execution of the patient care plan

30

35

302. Nursing orders 308 also generate corresponding interventions in the Kardex 310 and Task List 312.

Other interventions in the Kardex 310 and Task List 312 are mandated by hospital procedures 322, which include hospital standards of care, hospital policy, and regulations. The nurse or other hospital personnel 303 executes the orders as written in the Kardex 310 and Task List 312 and then charts on the appropriate forms to satisfy the requirement for providing necessary documentation.

Other forms such as Flowsheet 323 and Diagnostic Results 326 document specific measurements, tests results, and/or calculations 316 performed directly on the patient or on the patient's tissue, fluids, etc. Similarly, interventions and other therapy 320 are documented on a Medication Administration Record form 332 and other forms.

The nurse and physician observe the patient 315 and the results of interventions and therapy 320 and document their observations 318 in progress notes 324 and other associated forms.

From the observations 318 and progress notes 324 the physician 310 or nurse 303 writes a new set of orders 300 corresponding to the present condition of the patient 315.

System Hardware

Referring now to FIG. 2, a block diagram is shown of a typical hardware configuration for implementing the automated clinical records system of the present invention. FIG. 2 shows a distributed computer system comprising a plurality of workstations or terminals 2, 22, 32, 34 and 85 coupled to a local area network (LAN) 1.

The system is typically installed for use in a nursing care unit, such as an intensive care unit, in a hospital or clinic. Each of terminals 2 and 22 is located at the patient bedside. One terminal may be dedicated to the use of a single patient, or it may be used for multiple patients.

Terminals 32 and 34 may be located at a nursing station or nurse/physician lounge area. Terminal 85 is the system console which is used by a system administrator to configure and maintain the system and to provide additional services, such as displaying system status and error messages, archiving, and performing diagnostics.

Each bedside workstation or terminal, such as terminal 2, includes a video display unit with a viewable screen 3 for displaying information to the viewer; a housing 4 containing computing, data storage, and communications equipment and having associated with it a keyboard and pointing device such as a mouse 5; and connections 7 and 9 to one or more bedside devices 8 and 10. Bedside devices 8 and 10 may take the form of patient monitoring equipment suitable for the patient undergoing care, such as an EKG monitor, respiratory monitor, etc. Bedside terminal 22 may be coupled to a different set of bedside devices 28 and 30 from those coupled to terminal 2.

The nursing station or lounge terminals 32 and 34, and system console 85, may be identical to those used in the patient care unit but without the bedside device connections, or they may comprise slightly different equipment (e.g. personal computers) so long as they provide similar functionality.

Also coupled to the LAN 1 is a file server 62 and associated disc storage device 64. The file server 62 provides controlled access by the workstations 2, 22, 32, 34, and 85 to write information to and read information from disc storage device 64.

Optionally coupled to LAN 1 may be interfaces (not shown) to couple various system peripheral equipment to the LAN 1. For example, remote access modems may be coupled to one of such interfaces to provide access to the system from remote terminals (not shown) located elsewhere in the hospital or located offsite, such as at a physician's residence. Remote access may also be employed

to diagnose system problems from an off-site facility. A laboratory system may be connected to interface to permit the communication of laboratory information between the laboratory system and the clinical management system. An order communication system may be coupled to an interface to permit orders to be communicated from the system to other hospital systems (e.g. pharmacy or laboratory) and *vice versa*. An archival storage device may be coupled to an interface to permit any information stored in the system to be safe-stored on suitable archival media, such as magnetic tapes or optical discs.

Printers 81 and 83 are coupled to file server 62 to allow patient information to be printed for the convenience of hospital personnel and to maintain a suitable legal record of all observations, orders, parameter readings, care plans, and other patient information regarding the monitored patients. Printers 81 and 83 may be any suitable printers such as, for example, laser printers or high speed dot matrix printers. A printer may optionally be coupled to the bedside terminal and/or the terminal at the nursing station or lounge, if desired.

In operation, the system user, typically a nurse or physician, conducts a dialog with the system through the use of the keyboard, mouse, or other appropriate means for entering information such as a light-pen, touch-pad, trackball, etc.

"Icons", screen-sensitive areas, or the equivalent, or any combination thereof which is appropriate to the end application, may also be provided. "Icons" are symbols displayed on the screen whose functions are defined for the user by the system in view of the current screen context, and which can thus be readily changed according to the immediate requirements of the user application. In the present invention icons are selected by the system user by moving a screen cursor with the mouse and "clicking" on the

icon, i.e., depressing the mouse button while the cursor overlies the icon.

The user provides information or queries to the system by means of the keyboard and/or pointing device(s), and he receives information from the system by means of information displayed on the screen and/or through audible signals which could include, in an alternative embodiment, speech synthesis.

10

Patient Flowsheet

FIG. 3 shows a Flowsheet information screen in the automated clinical records system of the present invention.

15

The Flowsheet screen shown in FIG. 3 shows Vital Signs information for patient Donald M. Jackson which has been recorded at approximately hourly intervals. For example, Temperature (row 167), Heart Rate (row 168), and Systolic Arterial Pressure and Diastolic Arterial Pressure (row 169) have been measured and recorded.

20

Forms Manager

25

FIG. 4 shows a block diagram illustrating the relationship of the forms manager to the application program and database. A forms manager 111 is used as an interface, or buffer, among and between the input, output, and system applications and databases.

30

An input device 112 is coupled to forms manager 111. Input device 112 is illustrated as a keyboard with a joy stick but may be any type of input device. In general, the communication is from input device 112 to forms manager 111. However, there may be a need for forms manager 111 to operate visual or acoustic devices on input device 111. For this reason, there may be some communication from forms manager 111 to input device 112.

35

Also coupled to forms manager 111 is an operator display module 113. Display 113 is typically a cathode ray tube, but may be any of numerous types of displays such as the various types of flat panel displays. Communication is generally from forms manager 111 to display 113.

However, a touch sensitive screen or other type of input device may be incorporated in the system requiring communication from display 113 to forms manager 111.

Forms manager 111 also interacts with the application programs 114 and the databases. The databases, for this particular example, are broken into a general database 115 and a specific database 116. It should be understood that while the database may be functionally separated, both may be contained in the same physical location.

In this type of electronic database, a cell in a flowsheet may have a form or report associated with it which expands on the information in the cell. This form may be comprised of various attributes and processing rules obtained from one or more object instances of one or more object classes.

An object instance is the instantiation of an object class. An object class is similar to a type (as used in programming languages) in that it defines a structure. The information for these flowsheet cells and the underlying forms comes from a database containing various patient records. Examples of forms and records are found in copending patent applications "A Method of Forming a Spreadsheet Display" and "Spreadsheet Cell having Multiple Data Fields".

In operation, objects are entered through input device 112 to the forms manager. This input is checked and stored in database 116 for the particular patient. When a form is to be displayed, forms manager 111 obtains a list of objects to be displayed. One source of objects is application program 114. The forms manager then retrieves the display tiles associated with the objects from database 115. The data to be placed in the tiles is then obtained from database

116. Using these inputs, forms manager 111 develops a form which is displayed on display 113.

Application Software Architecture

5

FIG. 5 shows a block diagram of the application software architecture of the present invention.

10 The application software architecture is a way to specify action-oriented applications. The forms manager 100 translates user actions into "events" and sends the events to the graph manager 101.

The graph manager 101 screens incoming events, takes care of calling DECL's at appropriate times, handles dialog node sequencing, and import/export data flow.

15 The user performs actions. The forms manager 100 generates events based on the user's actions. The graph manager 101 catches the events generated by the forms manager 101.

20 An application written within the application software architecture consists of a set of dialog nodes collected into graphs (and possibly sub-graphs) and graph table entries that describe the graph. Dialog nodes in the application use various attributes. Each attribute has associated with it a group of procedures, any of which is optional. These
25 procedures are defaults, edits, calculations, and lists, collectively known as DECL's.

30 DEFAULT procedures are indicated generally by bubble 102; EDIT procedures by bubble 103; CALCULATION procedures by bubble 105; CROSS-EDIT procedures by bubble 104; and LIST procedures by bubble 107.

Function Graphs

35 The coordinating logic of an application is expressed as a set of function graphs. A function graph is a set of nodes and arcs connected into a directed graph. A function graph

allows an application to be divided into a number of independent but cooperating modules. Each node on the graph has a well-defined function that it performs when invoked. (A node in a graph may represent a sub-graph.)

5 The application designer specifies applications in terms of these graphs, drawing from a library of existing nodes and possibly adding new nodes to the library.

Function graphs are defined by a set of triplets of the form: {source_node, arc, destination_node}. Each function
10 graph has at least one triplet of the form {GRPH_INIT, arc, destination_node}. GRPH_INIT is a "pseudo-node" that defines the entry points to the graph. Each of the arcs that come out of GRPH_INIT should match one of the arcs coming into the function graph from its parent graph.

15 FIGS. 6A and 6B illustrate two function graphs. FIG. 6A illustrates a graph entitled "f_adt", and FIG. 6B illustrates a graph entitled "f_index", corresponding to the procedural node 205 of the same name within graph 200 of FIG. 6A.

In graph 200, node f_index is "called" via arc INDEX.
20 In graph 210, an arc called INDEX links GRPH_INIT 211 with the first node 212 of the graph.

To exit a graph, a least one triplet of the form {source_node, arc, GRPH_EXIT} should exist. GRPH_EXIT is a
25 "pseudo-node" that defines the exit points of the graph. Each of the arcs coming into GRPH_EXIT should match one of the arcs leaving the function graph in its parent graph.

Graph_mgt is responsible for managing the execution of these function graphs. Each node returns a status value when it completes. *Graph_mgt* uses the status value to
30 determine which arc of the parent graph to follow, and therefore which node to invoke next. In addition, an invoked node is passed the name of the arc which invoked it, giving the node additional information to perform its processing. A node on a function graph may be one of
35 several node types, or it may be another function graph.

This allows function graphs to be nested as required by the application.

Dialog Nodes

5

Each node in a function graph is structured as an event processor. Events are generated as a result of an action, such as clicking a mouse button or changing the value of a field. The body of the node is an event loop, processing each incoming event in sequence, until the node exits. Each node has a set of events that it understands, with code segments that define processing for each event.

10

FIG. 7 illustrates the structure of an dialog node. Each node is structured as an event processor. The body of the node is an event loop, processing each incoming event in sequence, until the node is exited. Each node has a set of events that it understands, with a code segment that defines the processing for that event. The event processing falls into one or more of the following categories:

15

20

- Display and removal of forms
- Display of a form menu
- Enabling / disabling of user entry into form data fields
- Acceptance / rejection user selection of a form object
- Creation / Modification / Deletion of object instances in the database (within the context of an object action).
- Dispatching of transactions to the database, rollback of previously entered object actions.
- Generation of messages to other processes (e.g. opening a patient chart, requesting the printing of a report, etc).

25

30

35

In order to perform this processing, the event-handling code typically needs to query the database, either

for the currently selected object instance, or for other data in the database.

Events

5

Events come from two sources: incoming graph arcs to the node (the first event a node receives is the graph arc that caused the node to be invoked), and events generated by user dialog (i.e. events triggered by user actions). These events are divided into two classes: predefined events and application defined events. Predefined events are generated by system and/or user actions under well defined conditions. Every node must define processing for these events. Applications may also define events that are associated with form menu choices. An application-defined event is generated whenever its associated menu choice is selected by the user.

10

15

Graph Table

20

Function graphs are defined by entries in a table in the database (the "graph table"). The entries define the sequencing between graph nodes and between graphs. Each entry in the graph table contains a source node name, an arc name (event), a destination node name, and the name of the graph containing the source node. Thus, given a graph name, a source node, and an event, the graph manager can look up the name of the node to invoke next (the destination node).

25

30

There are three special node names that are reserved by the graph manager. Entries in the graph table with "graph_init" for the source node name are entry points into a graph. Entries with "graph_exit" for the destination node name are exit points from a graph.

35

The Graph Table appended hereto shows the entries for the two function graphs entitled "f_adt" and "f_index"

shown in FIGS. 6A and 6B, respectively. For example, in graph "f_adt" an INIT event links source node GRAPH_INIT 201 and destination node S_CENSUS 203.

5

Event Handling Within the Graph Manager

Out of all the events (predefined and application defined), the graph manager catches/screens only six:
10 DEFAULT, NEW, SIGN, OK, MORE, and CROSS_EDIT.

DEFAULT and CROSS_EDIT events are never passed on to the application dialog nodes; the others might be passed on after certain processing is completed.

15 These events and the resulting processing are described in the pseudo-code appearing in Appendix B.

Detailed Description

DECL's in General

20

In the context of the present invention, an "attribute" is the smallest piece of useful data in a data object (i.e. data field in the database).

25 There are certain processing tasks associated with each attribute that don't change. For example the rules for validating user input of a particular attribute are the same regardless of which application is allowing the user to enter the data.

30 Traditionally, libraries are built for such functions, and all applications must call the appropriate functions for the attributes they use. Each function is written to process one particular attribute. When a library function is changed, deleted, or added, all of the applications that use the function (i.e. use the attribute the function is associated
35 with) must be (correctly) modified, recompiled, and retested.

Removing responsibility for such routine tasks (or procedures) from the applications reduces errors, maintenance time, implementation time, and design time. The following describes a method for automatically performing routine procedures so that the applications don't have to deal with them. Also described is a way to reduce the number of routine procedures required in a system.

5 An application uses various object attributes. Each attribute has associated with it a group of procedures, any of which is optional. These procedures may include defaults (supplies a default value for an attribute), edits (validates a new value for an attribute), calculations (calculates an attribute value from other attribute values), and lists (builds a sublist from a list). Defaults, edits, calculations, and lists are collectively known as DECL's.

10 In order to cut down on the number of different DECL's procedures in a system (in particular, edits), a DECL may be associated with more than one attribute. For example an edit procedure might exist that verifies that a date is not later than the current date. It would be better to associate the one edit with many attributes (e.g., admission date, birth date) than to write many edits that do the same thing, but are slightly different in that each deals with a different attribute.

20 In addition, edit procedures may be associated with an attribute type, so that an edit procedure is automatically applied to all attributes of a type and all the attributes of the type's children types. For example an edit might exist that verifies that a number entered by a user contains only digits and is in a particular range (e.g., 0-32767). This edit could be associated with all attributes whose type is *integer*. Further, the edit might be applied to attributes of type *int_id* (whose parent type is *integer*) and *staff_id* (whose parent type is *int_id*).

30 A type's "children" type is one that contains the characteristics of another type, referred to as a "parent"

type. For example, with reference to the Attribute Type Table appended hereto, the type "staff_id" is a "child" of the type "int_id". Through the concept of attribute type inheritance, all children must inherit the edit rules and storage format of their parent.

When an application uses an attribute, it automatically uses the DECL's associated with that attribute. Thus the application designer doesn't need to worry about the details handled by the DECL's, because the application doesn't have to deal with them. Once the DECL's are written for an attribute, any application that uses that attribute also uses those procedures (automatically).

Attributes may have any combinations of DECL's associated with them. For example, an attribute may have no DECL's, or it might have a list procedure and an edit procedure, but no transforms, defaults, or calculations.

Defaults

A default procedure determines the default value for its associated attribute, if such a value exists. An attribute may have at most one default procedure associated with it.

FIG. 8 illustrates the structure of a default procedure. Default procedures are invoked when a DEFAULT event is received for the associated field, based on a *forms_mgt* algorithm. A DEFAULT event occurs when the user selects a field for data entry. The default procedure determines the default value for the attribute, if one exists, and stores that value in the database.

Edits

An edit procedure defines a set of validity constraints, or edit rules, for one or more attributes. An edit rule contains an attribute specifier (since the edit procedure may be associated with more than one attribute which may be

5 "wild-carded" to include all attributes), a Boolean expression that represents the edit rule, and an error message to display if the edit rule fails. If the edit rule fails, the edit procedure exits (no subsequent edit rules within the edit procedure are tested).

10 In addition to edit rules, an edit procedure may define edit rule warnings. Edit rule warnings are the same as edit rules except that if the Boolean expression evaluates to false, the user is presented with a message and allowed to decide whether to proceed or not. If the user chooses to proceed, the edit rule passes. Otherwise it fails.

15 As mentioned above, each attribute may have zero or more edit procedures associated with it. In addition, attribute types may have edit procedures associated with them, which are applied to any attribute of that type. An attribute type also inherits the edits of any parent attribute types as defined in the system attribute type table.

20 The association between edit procedures and attribute types is specified in a database look-up table for attribute types. Refer to the Attribute Type Table appended hereto.

25 Attributes may have two classes of edits associated with them: single-field edits and cross-field edits. A single field edit is initiated whenever new data is entered for an attribute. The edit checks that the new value is valid based on either constant information (e.g., month must be between one and twelve, inclusive) or against "system data" (e.g., current date).

30 Cross-field edits are initiated for all fields that have received new data when the user tries to leave the form. They are also initiated for any fields designated "required" if the field has been enabled for entry (regardless of whether any data has been entered in the field). Cross-field edits are used to check consistency between the field they are associated with and other data in the database (e.g., that the
35 dose, concentration, and amount of an IV drip are

consistent). They are also used to implement required fields.

An attribute may have zero or more of each class of edit procedures.

5 FIG. 9 illustrates the structure of an edit procedure.

Calculations

10 A calculation procedure defines a way of deriving the value of an attribute based on other information in the database. A calculation procedure is initiated when a new value is entered into the system database for one of the "triggers" associated with the calculation procedure. A "trigger" is an independent variable in the equation that
15 calculates the target attribute. When the value of a trigger is updated, a calculation procedure for an associated attribute is called and performed.

A "calculation procedure" defines a way of deriving the value of an attribute based upon other information in the database. A calculation procedure may take the form of
20 an equation.

A trigger value may change in the system database, by action of a system user, for example, or by other appropriate input means such as the output of a transducer like a blood-pressure reader or urine collector.
25

The calculation procedure determines if it was invoked with a valid trigger, obtains the trigger data, does the calculation, and outputs the result. Outputting a result may trigger other dependent calculations.
30

FIG. 10 illustrates the structure of a calculation procedure.

Lists

35 A list procedure is used to implement a list whose list items vary during run-time. The list procedure selects

items from a master list based on criteria specified in the body of the list procedure and passes those items to the forms manager for display.

The association between DECL's and attributes is specified in a database look-up table for attributes. Refer to the Attribute Table appended hereto.

FIG. 11 illustrates the structure of a list procedure.

Attribute Order

Since so many DECL's may be associated with an attribute, the order in which they are applied to the attribute is important. For example edits are applied in the following order: All edits for the most primitive attribute type in the inheritance tree for the attribute's type, then the edits for the next child type, and so forth. Then all the edits for the attribute itself are applied. Within any given level, edits are applied in order of definition in the table (i.e. left to right).

Because cross-field edits can only be associated with attributes (not attribute types), only the cross-field edits for the specific attribute are invoked (in table definition order). An attribute must pass all of the edits associated with it in order to be considered valid.

Using the example tables in the appended Attribute Table: to be considered valid, the "who" attribute would have to pass (in order) edits *e_valid_int*, *e_valid_intid*, *e_valid_staffid*, *e_nurseid*, and *e_active_id*. The edit rule *e_valid_int* checks for a valid integer; *e_valid_intid* checks for a valid internal i.d.; *e_valid_staffid* checks for a valid staff i.d.; *e_nurseid* checks for a valid nurse i.d.; and *e_active_id* checks for a valid current i.d.

When specifying an application, the application designer must specify what attributes the application will use. When an application uses an attribute, it automatically uses the DECL's associated with that attribute. Thus the

application designer doesn't need to worry about the details handled by the DECL's, because the application doesn't have to deal with them. Once the DECL's are written for an attribute, any application that uses that attribute also uses those procedures (automatically).

It should again be emphasized that DECL's may be easily changed, in order to customize the system for a different user environment, simply by changing the appropriate attribute tables.

Attributes may have any combination of DECL's associated with them. For example, an attribute may have no DECL's, or it might have a list procedure and three edit procedures, but no defaults or calculations.

Applications are event-driven and are thus controlled by the process (or processes, as there may be more than one) they receive events from. Based on the events and the attribute(s) that are associated with them, the controlling process invokes the appropriate procedures associated with the attributes. Then, depending on the event and the results of the procedures, the controlling process optionally passes the event(s) to the application.

The controlling process determines which procedure(s) to invoke for a particular event and attribute by looking up the procedure in a table, such as Table 1. Thus to add, change, or delete a procedure merely requires changing appropriate table entries. The application code doesn't have to be changed.

DECL's and Application Software Architecture

There are two controlling processes, the forms manager and the graph manager. The forms manager handles all user I/O, as well as the details of exactly what appears on the screen, including such things as pop-up windows with lists of choices for the user. The forms

manager also translates user actions into events and sends those events to the graph manager.

5 The graph manager receives from the forms manager events and which attributes those events are associated with (one attribute per event). The graph manager screens incoming events, passing some to the applications and acting on others. Based on events received, the graph manager calls appropriate procedures to handle default values, validation of new values (edits), and calculation of values.

10 Both the forms manager and the graph manager determine which DECL's procedures to invoke by using a look-up table in a database.

15 The application sits underneath all of this waiting for events from the graph manager and dealing with them when received.

DECL's and the Graph Manager

20 When the graph manager receives a *default* event, it looks up in the attribute table the attribute for which the event was generated. If the attribute has a default procedure associated with it, the graph manager calls it.

25 The default procedure determines the default value for the attribute (possibly based on database attribute values, other screen values, etc), gives it to the forms manager to display on the screen, and returns.

30 The graph manager then looks up the calculation procedures that are *triggered* by the attribute (i.e. all calculations that depend on the attribute - for example 'B' is a trigger in the calculation "A=B+C") and calls the procedures, and any calculations that are triggered as a result. The graph manager doesn't pass *default* events to the application.

35 When the graph manager receives a *new* event, it looks up in the attribute table the attribute for which the event was generated. If the attribute has any edit

procedures (there may be zero or more) associated with it, the graph manager calls the edit procedures one at a time.

5 If any one edit procedure fails, the graph manager empties the event queue (where all incoming events are kept until the graph manager can process them). The edit procedure that failed notified the forms manager of the error and supplied the forms manager with an error message to display.

10 If all of the edit procedures for the attribute succeed, then the user has entered a valid value. The graph manager looks up the calculation procedures that are triggered by the attribute and calls the procedures, and any calculations that are triggered as a result.

15 After calling the calculation procedures (if any) for an attribute, the graph manager passes the *new* event to the application. If an edit fails, the *new* event is not passed to the application.

20 The graph manager passes all other events it receives directly to the application.

Lists and the Forms Manager

25 When the user selects a field on the screen that is enterable (i.e. the user is allowed to enter data), the graph manager looks up the field's associated attribute in an attribute table to determine if there is a list procedure associated with the attribute. If there is, the forms manager calls the list procedure, which returns a list of objects. The forms manager then displays the list in a pop-up window so
30 that the user can select an item from it for the selected field

The forms manager generates other special events, such as *cross-edit* events for cross-field edit procedures (which appear in the attribute table in the database under a column for cross-edit procedures).

DECL's - Detailed Implementation

Graph Management

5 FIG. 12 shows the "Graph Sequencing" function of
graph management. "Graph Sequencing" 230 is responsible
for managing the function graph aspect of graph
management and is initially started via a *CF-init-msg*. This
10 message includes the user function type that the *graph_mgt*
is executing on (e.g. CENSUS, CHART) as well as information
that is imported to the master function graph (e.g. PATIENT-
ID, WORKSTATION-NAME, etc.). "Graph Sequencing"
determines what the master graph is for this user function
15 type by looking in the *ENGINE-TABLE-D* and begins
executing it.

graph-d controls the sequencing of the graphs. A node
is invoked and based on *IMPORT-EXPORT-D*, any import
variables are passed to it (via *SC-node-invoke*). When the
node completes execution, the return status and *graph-d* are
20 used to determine the next node to invoke. When a GRPH-
EXIT is encountered in the master graph, graph sequencing
sits quietly and waits for the next *CF-init-msg*.

 FIG. 13 shows the "Manage Input Dialog" function of
graph management. "Manage Input Dialog" 231 is
25 responsible for controlling the response to user input events
(received via *CF-input-events*). Most events are passed
through to the application (via *CF-filtered-input-events*).

 DEFAULT and NEW trigger default or calculation
functions and single field edit processing (via *SC-perform-*
30 *defaults* and *SC-perform-edits*). DEFAULTs are never
passed through to the application. NEWS are passed to the
application for an entered field after any default/edit
processing and if it passes any edit processing.

 SIGN, OK and MORE trigger performance of cross-field
35 edit checks and are only passed through to the application if
all edits succeed.

When a DEFAULT or NEW event is received, "Manage Input Dialog" does an *FM-BEGIN-UPDATE* to hold any form updates (as a result of FM-UPDATES done by "Default-Proc". see FIG. 15). "Manage Input Dialog" issues an *FM-END-UPDATE* to allow those form updates to proceed after all the default/calculation processing has completed.

When commanded to perform a default/calculation on a specified attribute, "Control Default/Calcs" 233 invokes the appropriate "Default Proc" (known via *ATTRIBUTE-DEFAULT-D*) via *SC-default-invoke*. This causes any fields dependent on the newly defaulted/calculated field to be defaulted/calculated via *SC-default-invokes* (based on information in *ATTRIBUTE-DEFAULT-D*).

"Control Edits" 235, when commanded to perform single field edits on a specified attribute (by *SC-perform-edits*), causes all the associated edits (known from *FORM-EDIT-D*) on that attribute to be done via *SC-edit-invoke*. A failure of any will abort further edit processing for that attribute. If commanded to perform cross-field edits, all the edits for the currently active form (fetched via *FM-GET-CURRENT-FORM* and *CURRENT-FORM*) are performed (known via *FORM-EDIT-D*). *FORM-EDIT-D* contains two sets of information: a) all single field edits for the attributes on a form, and b) all cross-field edits on a form.

Single field edits for an attribute are invoked in the following order: All edits for the most primitive attribute type in the inheritance tree for the attribute's type, then the edits for the next child attribute type, and so forth. Then all edits for the attribute itself. Within a given level, edits are invoked in order of definition (i.e. left to right).

FIG. 14 shows a "Dialog Node" 241. Dialog nodes interact with the Form Manager to control data display and entry. When dialog node 241 is invoked (via *SC-node-invoke*), it retrieves the definition of its processing (via *QR-node-d*), and also receives a set of import attributes and the name of the invoking arc (via *SC-node-invoke*). It may use

a variety of tables to specify options in how it controls the dialog (via *QR-app-config-d*).

5 Dialog nodes make requests to the Form Manager to display data on various forms (via *CF-interaction-control* and *interaction-results*) and receive notification of various events (e.g. field selected) via *CF-filtered-input-events*. The dialog nodes specify what actions are to be carried out for each of these events (e.g. change menu definitions, proceed to a particular field), again via *CF-interaction-control*.

10 As a result of the user dialog, a dialog node may request object instance data (via *QR-object-data*), modify that object instance data (via *object-actions*) and cause those object actions to be sent to the database (via *SC-mr-ed-control*). It may also generate a variety of system control
15 commands (via *runtime-sys-control*).

FIG. 15 shows a "Default Procedures" node 242. Defaults are a way of loading a value into a field on a form "automatically". The algorithm for determining the default value (the default "rule") is specified in a "Default
20 Procedure". A calculation is also implemented via a default rule.

Graph_mgt invokes a default procedure via *SC-default-invoke*. The description of the default rules is then retrieved via *QR-default-d*. A set of information is passed in
25 the invoke that enables the procedure to determine for what kind of object and field it has been invoked. The default procedure then determines the default value. It may utilize other previously stored data to do so (via *QR-object-data*), or it may use unvalidated monitor data (via *monitor-db*).
30 Information obtained from *QR-app-config-d* may be used to determine any options. Once it has the default value, it is entered into the database via *object-actions*. The form is refreshed via *FM-UPDATE*.

FIG. 16 shows a "List Procedures" node 243. When a
35 field (attribute) is selected by the user on a form, if it has a list, the *form_mgt* will cause it to be displayed. Lists are

associated with the attribute. Some attributes have fixed lists, which are handled by *form_mgt*. Other attributes have dynamic lists that change based on the value of other attributes previously entered. List procedures are responsible for creating those dynamic lists.

The *form_mgt* invokes the list procedure via *SC-list-invoke*. The list procedure then retrieves a description of its rules via *QR-list-d*. It may use information obtained via *QR-app-config-d* or *QR-object-data* to control options. It then determines what the contents of the list to display are and outputs the list via *list-contents*.

FIG. 17 shows a "Edit Procedures" node 244. Edit procedures are defined to establish validity constraints for attributes. The rules for determining if an attribute passes those validity constraints are embodied in an edit procedure. Each attribute may have zero to N edit procedures assigned to it. An attribute must pass ALL of the edits in order to be valid.

Graph_mgt invokes an edit procedure via *SC-edit-invoke*. The description of the edit rule is then retrieved via *QR-edit-d*. A set of information is passed in the invoke that enables the procedure to determine for what kind of object and field it has been invoked (and whether it is being invoked for a single field or a cross-field edit). The edit procedure then determines if the value of the attribute is valid. It may utilize other previously stored data to do so (via *QR-object-data*). Information obtained from *QR-app-config-d* may be used to determine any options.

If the value is valid, *SC-edit-invoke* is completed with an ACCEPT. If the field is invalid (and in single-field edit mode), the edit procedure forces a selection of the erroneous field via *FM-DO-SELECT*, flags the field in error via *FM-FIELD-IN-ERROR* and then *SC-edit-invoke* is completed with a reject.

Before returning, the edit procedure has the option of displaying information to the user. It may use *FM-LOCAL-*

MSG to show advisory information that does not have to be acknowledged. It may use *FM-LOCAL-ERROR* to display information that has to be acknowledged before the user can proceed. If in cross-field edit mode, the edit procedure should only flag the field in error (via *FM-FIELD-IN-ERROR*) and complete *SC-edit-invoke* with a reject.

FIGS. 18A-18D show a flow diagram corresponding to the pseudo-code of Appendix B, illustrating how DECL's functions are associated with attributes according to a preferred embodiment of the invention.

In FIG. 18A incoming events enter decision box 500; if the event is a DEFAULT, it proceeds to Routine A in FIG. 18B and ultimately returns via path 501 to handle the next incoming event; if not, it continues with decision box 502. In box 502, if the event is a NEW, it proceeds to Routine B in FIG. 18C and ultimately returns via path 503 to enter box 510; if not, it continues with decision box 504. In decision box 504, if the event is a CROSS-EDIT, it proceeds to Routine C in FIG. 18D and ultimately returns via path 501 to handle the next incoming event; if not, it continues with decision box 506.

In decision box 506, if the event is an OK, MORE, or SIGN, it proceeds to decision box 508, where if the Edit_Failed flag is set, the procedure returns via path 501 to handle the next incoming event, but if not it proceeds to box 510, where the event is passed on to the dialog node. In decision box 506, if the event is not an OK, MORE, or SIGN, it proceeds to box 510 and then the procedure returns via path 501 to handle the next incoming event.

In FIG. 18B, Routine A begins by entering box 512, where the DEFAULT procedure is looked up in the Attribute Table for the current attribute. Next the routine enters decision box 514, where if the DEFAULT procedure does not exist for the attribute, the routine exits, and if it does exist, the routine proceeds to box 516. In box 516, the DEFAULT procedure is invoked, and then Routine A exits.

In FIG. 18C, Routine B begins by entering box 518, where the current field's attribute type and rules are looked up in the Attribute Table. Next in box 520 the EDIT rules for the attribute type and its parents are looked up in the Attribute Type Table.

5

Then the routine proceeds to loop box 524, where if more EDIT rules are on the table list, then it proceeds to box 526 where the EDIT rule is invoked and the routine goes to decision box 522. In decision box 522, if the EDIT rule passed, the routine returns to the loop box 524, but if not it exits.

10

When no more EDIT rules are on the list, the routine leaves the EDIT rules loop and goes to box 528, where the CALC rules for the current attribute are looked up in the Attribute Table.

15

Next the routine proceeds to loop box 530, where if more CALC rules are on the table list, it proceeds to box 532, where the CALC rule is invoked, and then returns to the loop box 530. When no more CALC rules are on the list, the routine exits.

20

In FIG. 18D, Routine C begins by entering box 534 where the Edit_Failed flag is cleared. Next in box 536, the current field's CROSS-EDIT rules are looked up in the Attribute Table, and the routine proceeds to loop box 538.

25

In loop box 538, if there are more CROSS-EDIT rules on the list, the routine proceeds to box 540; but if not it exits. In box 540 the particular CROSS-EDIT rule is invoked.

30

Then the routine goes to decision box 542; if the CROSS-EDIT rule passed, the routine returns to enter loop box 538, but if not it goes to box 544, where the Edit_Failed flag is set, whereupon Routine C exits.

35

It will be apparent to those skilled in the art that the disclosed invention may be modified in numerous ways and may assume many embodiments other than the preferred form specifically set out and described above.

For example, the division of functionality between the graph manager and the forms manager may be altered in different ways, so that one may perform functions described herein as performed by the other, and *vice versa*.

5

Accordingly, it is intended by the appended claims to cover all modifications of the invention which fall within the true spirit and scope of the invention.

What is claimed is:

Graph Name	Source Node	Event	Destination Node
f_adt	graph_init	INIT	s_census
f_adt	s_census	INDEX	f_index
f_adt	f_index	CENSUS	s_census
f_adt	f_index	READMIT	s_census
f_adt	null_node	SHUTDOWN	graph_exit
f_index	graph_init	INDEX	s_index
f_index	s_index	SHOWDETAIL	s_demo
f_index	s_index	REVIEWCHART	n_openchart
f_index	s_index	ADMIT	s_adt
f_index	s_index	CENSUS	graph_exit
f_index	s_demo	RETURN	s_index
f_index	n_openchart	RETURN	s_index
f_index	s_adt	RETURN	s_index
f_index	s_adt	READMIT	graph_exit

GRAPH TABLE

Attr Name	Attr Type	Default Rule	List Rule	Calc Rule	Edit-sf Rule(s)	Edit-cf Rule(s)
who	STAFFID	d_cur_id			e_nurseid, e_activeid e_bdate	
bdate	DATE					
age	AGE_T	e_cf_age				
io_sum	IO_SUM			c_ios(io_in, io_out)		
dname	NAME_T		l_docs		e_docname,	
	e_cf_docid					
dr_id	STAFFID		l_docids		e_active_d e_docid	
	e_cf_docn					
					e_active_d	

ATTRIBUTE TABLE

Attr Type	Parent Type	Edit
STAFFID	INTID	e_valid_staffid
INTID	integer	e_valid_intid
DATE	long	e_valid_date
AGE_T	integer	e_valid_age
IO_SUM	float	
NAME_T	char*	

ATTRIBUTE TYPE TABLE

APPENDIX A

GLOSSARY

Arc = an event, represented graphically as a line into or out of a node.

Attribute = a field within an object class, e.g. patient i.d.

Attribute name = an identifier by which the attribute is selected.

Attribute table = a table containing the definition of every attribute in the entire system, including the associated attribute name, calculation procedure, default procedure, edit procedure (single-field edits, cross-field edits), list procedure, transform procedures, and type name.

Attribute type = defines physical characteristics of an attribute.

Blackboard = an area of memory reserved for use by the nodes of the current directed graph. All of a graph's imports and exports are stored in its blackboard until the procedure exits from the graph.

DECL's = Default, Edit, Calculation, and List procedures. These are associated with attributes, not with specific applications. An attribute may have any combination of DECL's (or no DECL's) associated with it.

Calculation procedure = defines a way of deriving the value of an attribute based on other information in the database. Initiated when a value is entered into the database (by the user or otherwise) for one of the triggers associated with the calculation procedure.

Default procedure = loads a value into a field on a form automatically when the field has been selected for entry. The default procedure determines the default value, if any, for an attribute.

Edit procedure = defines a set of validity constraints, or edit rules, for one or more attributes. An edit rule contains an attribute specifier (since an edit procedure may be associated with more than one attribute), a Boolean expression that represents the edit rule, and an error message to display if the edit rule fails.

Single-field edits = initiated whenever new data is entered for the attribute on a form.

Cross-field edits = initiated for any field which has received new data on a form whenever the OK, MORE, or SIGN actions are selected by the user. It is also initiated for any fields designated in the form definition as REQUIRED if that field has been enabled for entry. They are used to check consistency between the field they are associated with and other data in the database (e.g. that the dose, concentration, and amount of an IV drip are consistent).

List procedure = used to select a subset of list elements, based upon some algorithm, from a superset of possible list elements. The superset is defined in a list table. A list procedure is invoked whenever the associated field is selected and enterable.

Dialog node = interacts with the forms manager to control data display and entry. Makes requests to the forms manager to display data on various forms and receive notification of various events.

Event = generated as a result of an action, either triggered by user action or as an incoming graph arc to the node.

Export = individual node or graph can export data.

Forms manager = translates user actions into events and sends those events to the graph manager.

Graph = the coordinating logic of an application is expressed as a set of function graphs.

Directed graph = same as a function graph.

Exit graph = the last node in a graph, it copies its graph's export values from its blackboard to the calling graph's blackboard and pops the blackboard stack. It handles a graph's export attributes.

Function graph = a set of nodes and arcs connected into a directed graph. Permits an application to be divided into a number of independent but cooperating parts. Each node on the graph has a well-defined function that it performs when invoked.

Graph manager = coordinates execution of an application expressed as a directed graph, including screening incoming events, calling DECL's at appropriate times, handles dialog node sequencing, and import/export data flow.

Graph table = entries in a table in a database that describe a function graph. They describe the sequencing

between graph nodes and between graphs. Each entry in the graph table contains a source node name, an arc name (event), a destination node name, and the name of the graph containing the source node.

Initialization graph = the first node in a graph, it is a graph manager function that sets up the current graph's blackboard. When invoked, it creates a blackboard, copies the graph's imports onto the blackboard, and pushes the blackboard onto the blackboard stack. It handles a graph's import attributes.

Nested graphs = a graph containing a sub-graph.

Sub-graph = a graph comprising two or more nodes contained in a node of a graph.

Import = individual node or graph can import data.

List = a set of predefined choices for entry into a field.

MORE = the user selects the "MORE" icon on the form to request more information to be retrieved from the system for display on the form.

NEW = the user enters new data into a field.

Node = A portion of a function graph that performs a well-defined function when invoked. Nodes are interconnected by arcs representing events. Nodes specify what actions are to be carried out for each of these events. Each node in a function graph is structured as an event processor. The body of the node is an event loop, processing each incoming event in sequence, until the loop exits.

Object action = a well-defined set of operations that modifies the data structures associated with an object instance to reflect an event in the real world (e.g. create a physician's order; chart a medication).

Object class = a logical collection of data attributes which describe a class of items in the real world (outside of the system) and which are treated as a unit; e.g. Vital Signs (includes all related objects, such as Respiratory Rate, Blood Pressure, Heart Rate, Reflexes, etc.).

Object instance = each physical item that is being tracked by the system is represented by an instantiation of the appropriate object class.

OK = the user indicates he/she is done with this particular pop-up form.

SIGN = the user commits to the data he/she has entered.

Trigger = one of the independent variables in the equation that calculates the target attribute.

APPENDIX B

PSEUDO-CODE

Event Loop (Graph Mgr)

Case DEFAULT:

In attribute table, look up the default proc for the current attribute.

If the default proc exists, call it.

Break (next event).

Case NEW:

In the attribute table, look up the current field's attribute type and edit rules.

In the attribute type table, look up the current field's attribute type and that of all its parents, and get all the edit rules.

Call all the edit rules found in order.

If any one of the edit rules fails
Break (next event).

Else (all edits passed)

In the attribute table, look up the calculation rules triggered by the current attribute.

For each such calculation rule, call it and any calculation it triggers.

Pass the NEW event on to the current dialog node.

Break (next event).

39 Case CROSS_EDIT:
40 In the attribute table, look up the current field's
41 cross_edit rules.
42
43 Call all the cross_edit rules found in order.
44
45 If any one of the cross_edit rules fails
46 Break (next event).
47
48 Break (next event).
49
50 Case OK:
51 Case SIGN:
52 Case MORE:
53 If any of the previous edits failed (this takes care of
54 case where forms manager sends a slew of cross_edit
55 events followed by an ok, sign, or more event)
56 Ignore this event
57
58 Else
59 Pass the event on to the dialog node.
60
61 Break (next event).
62
63 Default:
64 Pass the event on to the dialog node.
65 Break (next event).
66

THE EMBODIMENT OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. A method performed by a computer with an input means, the method comprising the steps of:
 - (a) storing an attribute identifier, an attribute-specific procedure identifier and an attribute type identifier in an attribute table entry in an attribute table;
 - (b) storing the attribute type identifier, a type-specific procedure identifier and a parent type identifier in a first type table entry in a type table;
 - (c) storing the parent type identifier as a second attribute type identifier and a parent type-specific procedure identifier as a second typespecific procedure identifier in a second type table entry in the type table;
 - (d) executing an application program that obtains data entered via the input means; and
 - (e) executing a separate program upon determining that the entered data in step (d) is an attribute having the attribute identifier stored in the attribute table, the execution of the separate program including the steps of:
 - (e1) locating the attribute identifier in the attribute table;
 - (e2) determining the attribute type identifier stored in the attribute table entry in the attribute table that stores the attribute identifier located in step (e1);
 - (e3) determining the parent type identifier in the first type table entry that stores the attribute type identifier;
 - (e4) executing a parent type-specific procedure identified by the parent type-specific procedure identifier stored as the second type-specific procedure identifier in the second type table entry that stores the parent type identifier determined in step (e3) as the second attribute type identifier;
 - (e5) executing a type-specific procedure identified by the typespecific procedure identifier stored in the first type table entry that stores the attribute type identifier determined in step (e2); and

- (e6) executing an attribute-specific procedure identified by the attribute-specific procedure identifier stored in the attribute table entry in the attribute table that stores the attribute identifier located in step (e1).

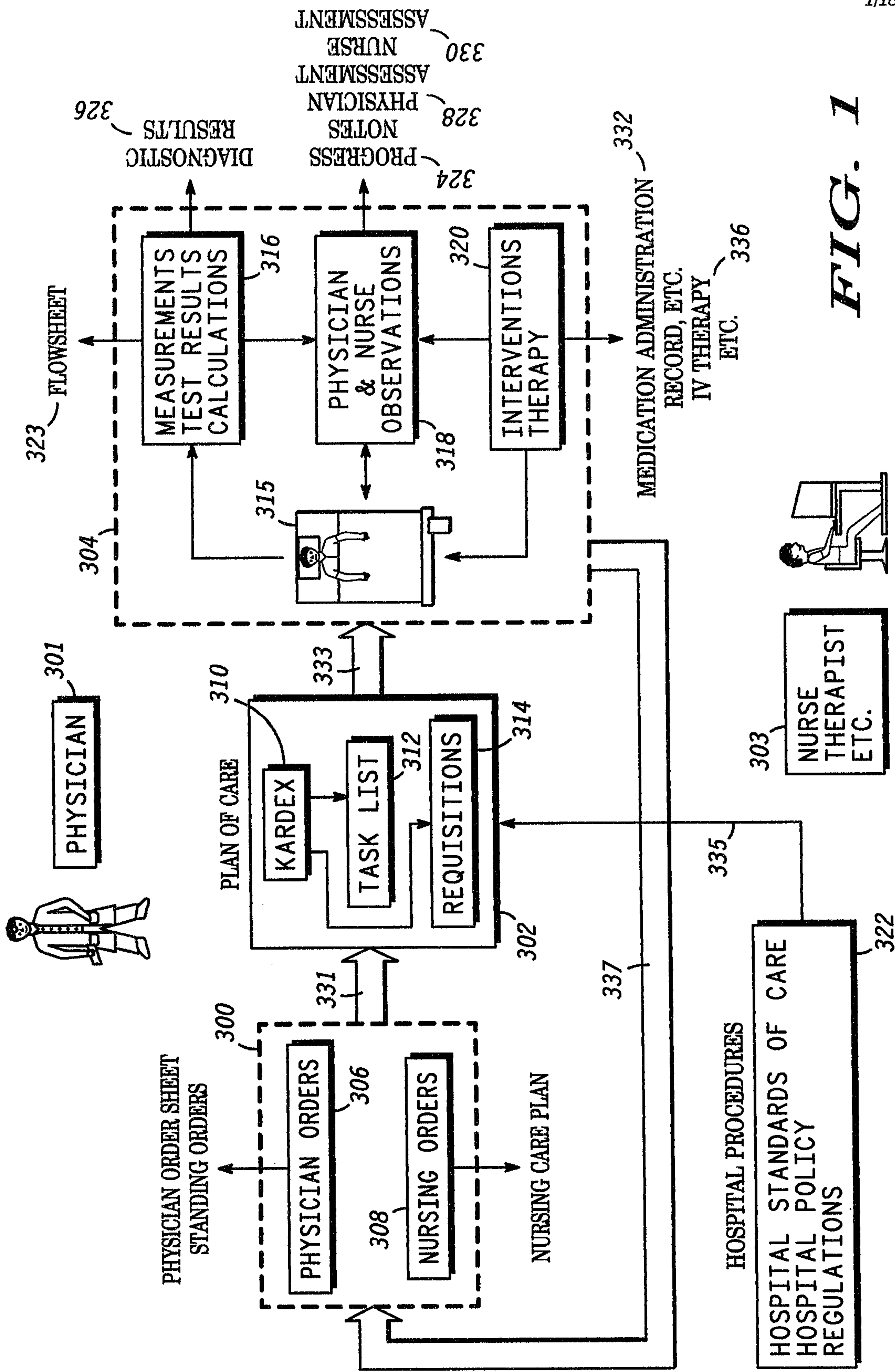


FIG. 1

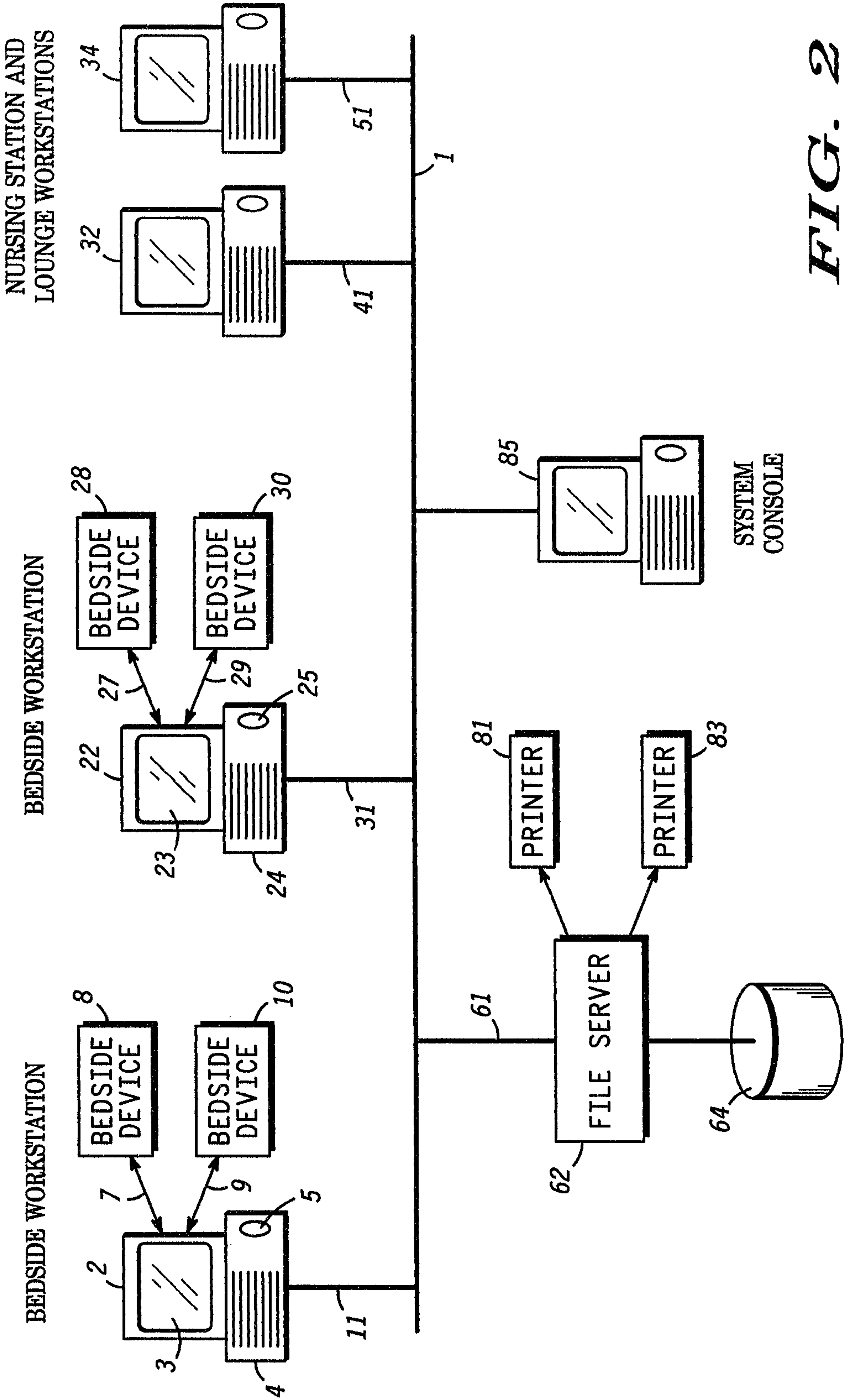


FIG. 2

150

METRO MEDICAL CENTER

152

BLOOD GAS RESULTS READY

154

SECTIONS: [FLOWSHEET] ORDERS NCP ASSESSMENT LABS R.T. KARDEX

156

FORMS: MEDS [VITALS] I&O LABS VENTILATOR

158

01/07 07:00 08:00 09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00

160

TEMP

162

98.6

164

90

166

80

168

HEART RATE

170

100

172

60

174

105

176

65

178

BP

180

100

182

60

184

120

186

80

188

150

190

60

192

100

194

80

196

100

198

85

200

100

202

80

204

100

206

80

208

100

210

80

212

100

214

80

216

100

218

80

220

100

222

80

224

100

226

80

228

100

230

80

232

100

234

80

236

100

238

80

240

100

242

80

244

100

246

80

248

100

250

80

252

100

254

80

256

100

258

80

260

100

262

80

264

100

266

80

268

100

270

80

272

100

274

80

276

100

278

80

280

100

282

80

284

100

286

80

288

100

290

80

292

100

294

80

296

100

298

80

300

100

302

80

304

100

306

80

308

100

310

80

312

100

314

80

316

100

318

80

320

100

322

80

324

100

326

80

328

100

330

80

332

100

334

80

336

100

338

80

340

100

342

80

344

100

346

80

348

100

350

80

352

100

354

80

356

100

358

80

360

100

362

80

364

100

366

80

368

100

370

80

372

100

374

80

376

100

378

80

380

100

382

80

384

100

386

80

388

100

390

80

392

100

394

80

396

100

398

80

400

100

402

80

404

100

406

80

408

100

410

80

412

100

414

80

416

100

418

80

420

100

422

80

424

100

426

80

428

100

430

80

432

100

434

80

436

100

438

80

440

100

442

80

444

100

446

80

448

100

450

80

452

100

454

80

456

100

458

80

460

100

462

80

464

100

466

80

468

100

470

80

472

100

474

80

476

100

478

80

480

100

482

80

484

100

486

80

488

100

490

80

492

100

494

80

496

100

498

80

500

100

502

80

504

100

506

80

508

100

510

80

512

100

514

80

516

100

518

80

520

100

522

80

524

100

526

80

528

100

530

80

532

100

534

80

536

100

538

80

540

100

542

80

544

100

546

80

548

100

550

80

552

100

554

80

556

100

558

80

560

100

562

80

564

100

566

80

568

100

570

80

572

100

574

80

576

100

578

80

580

100

582

80

584

100

586

80

588

100

590

80

592

100

594

80

596

100

598

80

600

100

602

80

604

100

606

80

608

100

610

80

612

100

614

80

616

100

618

80

620

100

622

80

624

100

626

80

628

100

630

80

632

100

634

80

636

100

638

80

640

100

642

80

644

100

646

80

648

100

650

80

652

100

654

80

656

100

658

80

660

100

662

80

664

100

666

80

668

100

670

80

672

100

674

80

676

100

678

80

680

100

682

80

684

100

686

80

688

100

690

80

692

100

694

80

696

100

698

80

700

100

702

80

704

100

706

80

708

100

710

80

712

100

714

80

716

100

718

80

720

100

722

80

724

100

726

80

728

100

730

80

732

100

734

80

736

100

738

80

740

100

742

80

744

100

746

80

748

100

750

80

752

100

754

80

756

100

758

80

760

100

762

80

764

100

766

80

768

100

770

80

772

100

774

80

776

100

778

80

780

100

782

80

784

100

786

80

788

100

790

80

792

100

794

80

796

100

798

80

800

100

802

80

804

100

806

80

808

100

810

80

812

100

814

80

816

100

818

80

820

100

822

80

824

100

826

80

828

100

830

80

832

100

834

80

836

100

838

80

840

100

842

80

844

100

846

80

848

100

850

80

852

100

854

80

856

100

858

80

860

100

862

80

864

100

866

80

868

100

870

80

872

100

874

80

876

100

878

80

880

100

882

80

884

100

886

80

888

100

890

80

892

100

894

80

896

100

898

80

900

100

902

80

904

100

906

80

908

100

910

80

912

100

914

80

916

100

918

80

920

100

922

80

924

100

926

80

928

100

930

80

932

100

934

80

936

100

938

80

940

100

942

80

944

100

946

80

948

100

950

80

952

100

954

80

956

100

958

80

960

100

962

80

964

100

966

80

968

100

970

80

972

100

974

80

976

100

978

80

980

100

982

80

984

100

986

80

988

100

990

80

992

100

994

80

996

100

998

80

1000

100

1002

80

1004

100

1006

80

1008

100

1010

80

1012

100

1014

80

1016

100

1018

80

1020

100

1022

80

1024

100

1026

80

1028

100

1030

80

1032

100

1034

80

1036

100

1038

80

1040

100

1042

80

1044

100

1046

80

1048

100

1050

80

1052

100

1054

80

1056

100

1058

80

1060

100

1062

80

1064

100

1066

80

1068

100

1070

80

1072

100

1074

80

1076

100

1078

80

1080

100

1082

80

1084

100

1086

80

1088

100

1090

80

1092

100

1094

80

1096

100

1098

80

1100

100

1102

80

1104

100

1106

80

1108

100

1110

80

1112

100

1114

80

1116

100

1118

80

1120

100

1122

80

1124

100

1126

80

1128

100

1130

80

1132

100

1134

80

1136

100

1138

80

1140

100

1142

80

1144

100

1146

80

1148

100

1150

80

1152

100

1154

80

1156

100

1158

80

1160

100

1162

80

1164

100

1166

80

1168

100

1170

80

1172

100

1174

80

1176

100

1178

80

1180

100

1182

80

1184

100

1186

80

1188

100

1190

80

1192

100

1194

80

1196

100

1198

80

1200

100

1202

80

1204

100

1206

80

1208

100

1210

80

1212

100

1214

80

1216

100

1218

80

1220

100

1222

80

1224

100

1226

80

1228

100

1230

80

1232

100

1234

80

1236

100

1238

80

1240

100

1242

80

1244

100

1246

80

1248

100

1250

80

1252

100

1254

80

1256

100

1258

80

1260

100

1262

80

1264

100

1266

80

1268

100

1270

80

1272

100

1274

80

1276

100

1278

80

1280

100

1282

80

1284

100

1286

80

1288

100

1290

80

1292

100

1294

80

1296

100

1298

80

1300

100

1302

80

1304

100

1306

80

1308

100

1310

80

1312

100

1314

80

1316

100

1318

80

1320

100

1322

80

1324

100

1326

80

1328

100

1330

80

1332

100

1334

80

1336

100

1338

80

1340

100

1342

80

1344

100

1346

80

1348

100

1350

80

1352

100

1354

80

1356

100

1358

80

1360

100

1362

80

1364

100

1366

80

1368

100

1370

80

1372

100

1374

80

1376

100

1378

80

1380

100

1382

80

1384

100

1386

80

1388

100

1390

80

1392

100

1394

80

1396

100

1398

80

1400

100

1402

80

1404

100

1406

80

1408

100

1410

80

1412

100

1414

80

1416

100

1418

80

1420

100

1422

80

1424

100

1426

80

1428

100

1430

80

1432

100

1434

80

1436

100

1438

80

1440

100

1442

80

1444

100

1446

80

1448

100

1450

80

1452

100

1454

80

1456

100

1458

80

1460

100

1462

80

1464

100

1466

80

1468

100

1470

80

1472

100

1474

80

1476

100

1478

80

1480

100

1482

80

1484

100

1486

80

1488

100

1490

80

1492

100

1494

80

1496

100

1498

80

1500

100

1502

80

1504

100

1506

80

1508

100

1510

80

1512

100

1514

80

1516

100

1518

80

1520

100

1522

80

1524

100

1526

80

1528

100

1530

80

1532

100

1534

80

1536

100

1538

80

1540

100

1542

80

1544

100

1546

80

1548

100

1550

80

1552

100

1554

80

1556

100

1558

80

1560

100

1562

80

1564

100

1566

80

1568

100

1570

80

1572

100

1574

80

1576

100

1578

80

1580

100

1582

80

1584

100

1586

80

1588

100

1590

80

1592

100

1594

80

1596

100

1598

80

1600

100

1602

80

1604

100

1606

80

1608

100

1610

80

1612

100

1614

80

1616

100

1618

80

1620

100

1622

80

1624

100

1626

80

1628

100

1630

80

1632

100

1634

80

1636

100

1638

80

1640

100

1642

80

1644

100

1646

80

1648

100

1650

80

1652

100

1654

80

1656

100

1658

80

1660

100

1662

80

1664

100

1666

80

1668

100

1670

80

1672

100

1674

80

1676

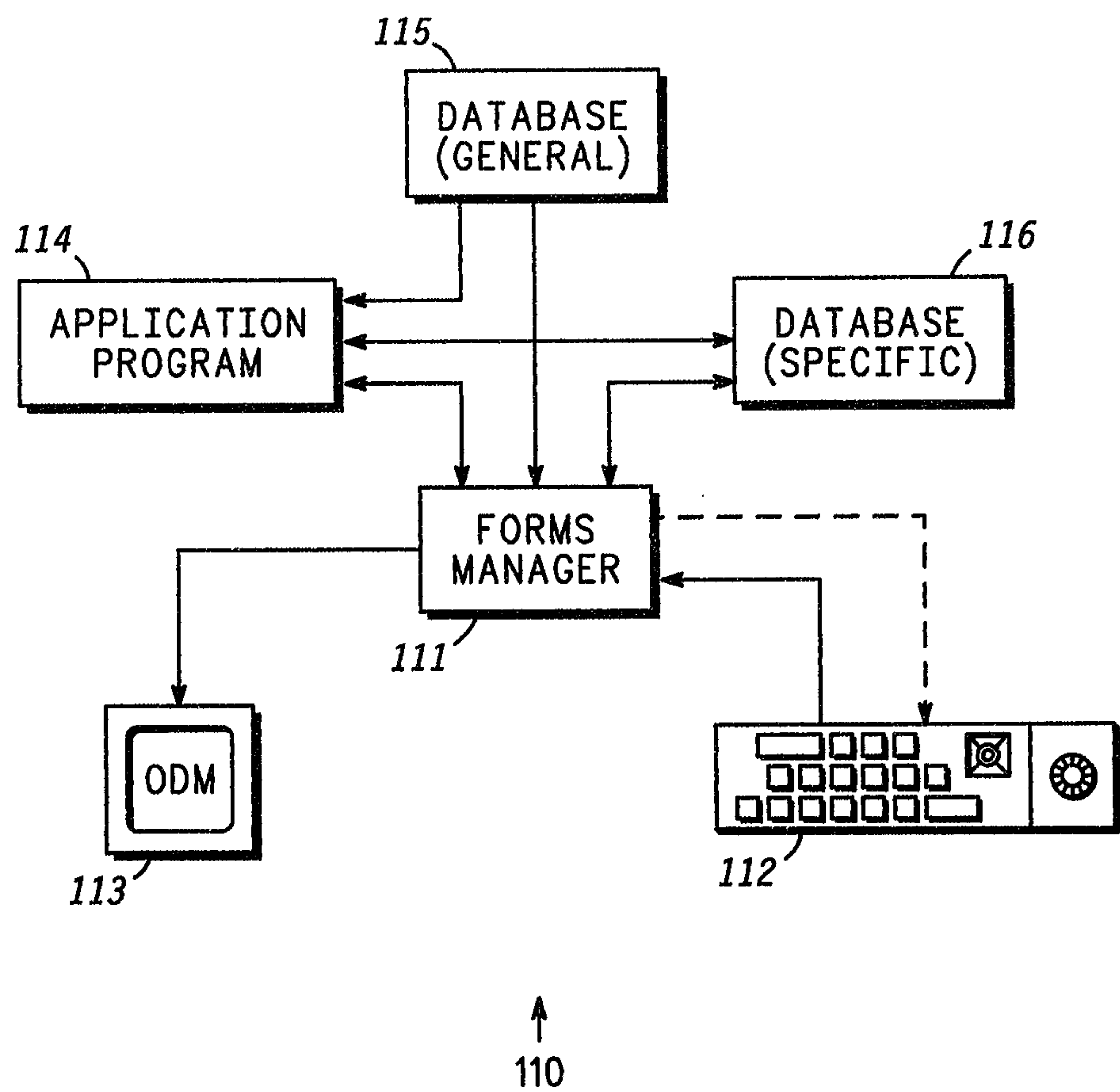
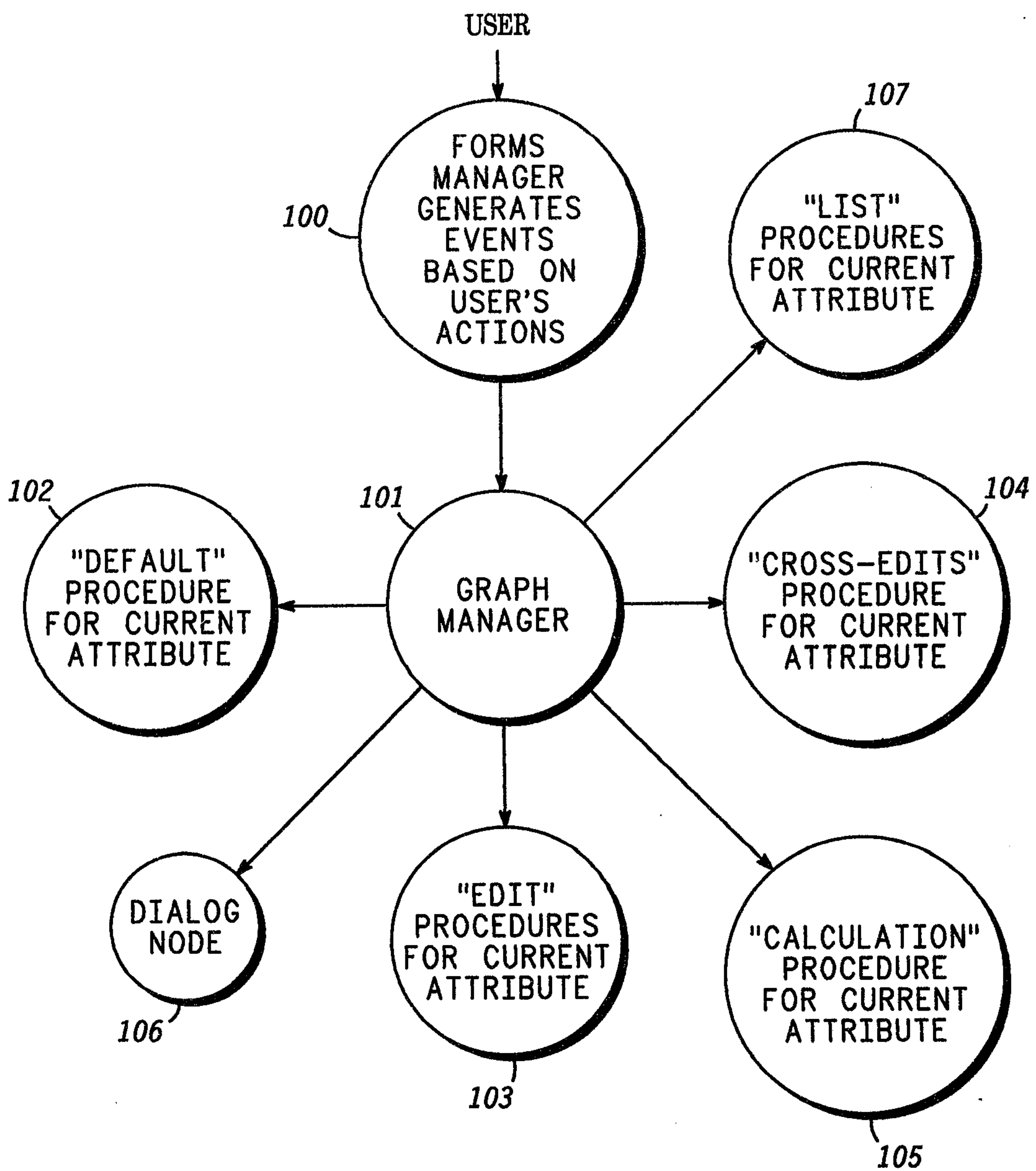


FIG. 4

5/18

**FIG. 5**

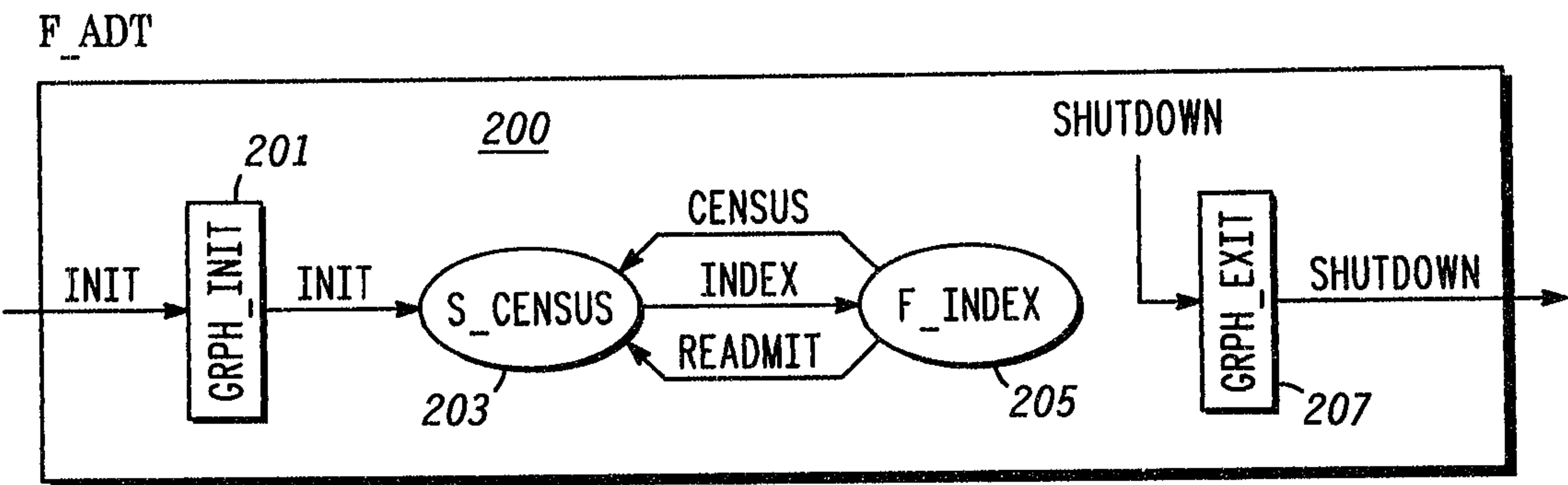


FIG. 6A

FIG. 6B

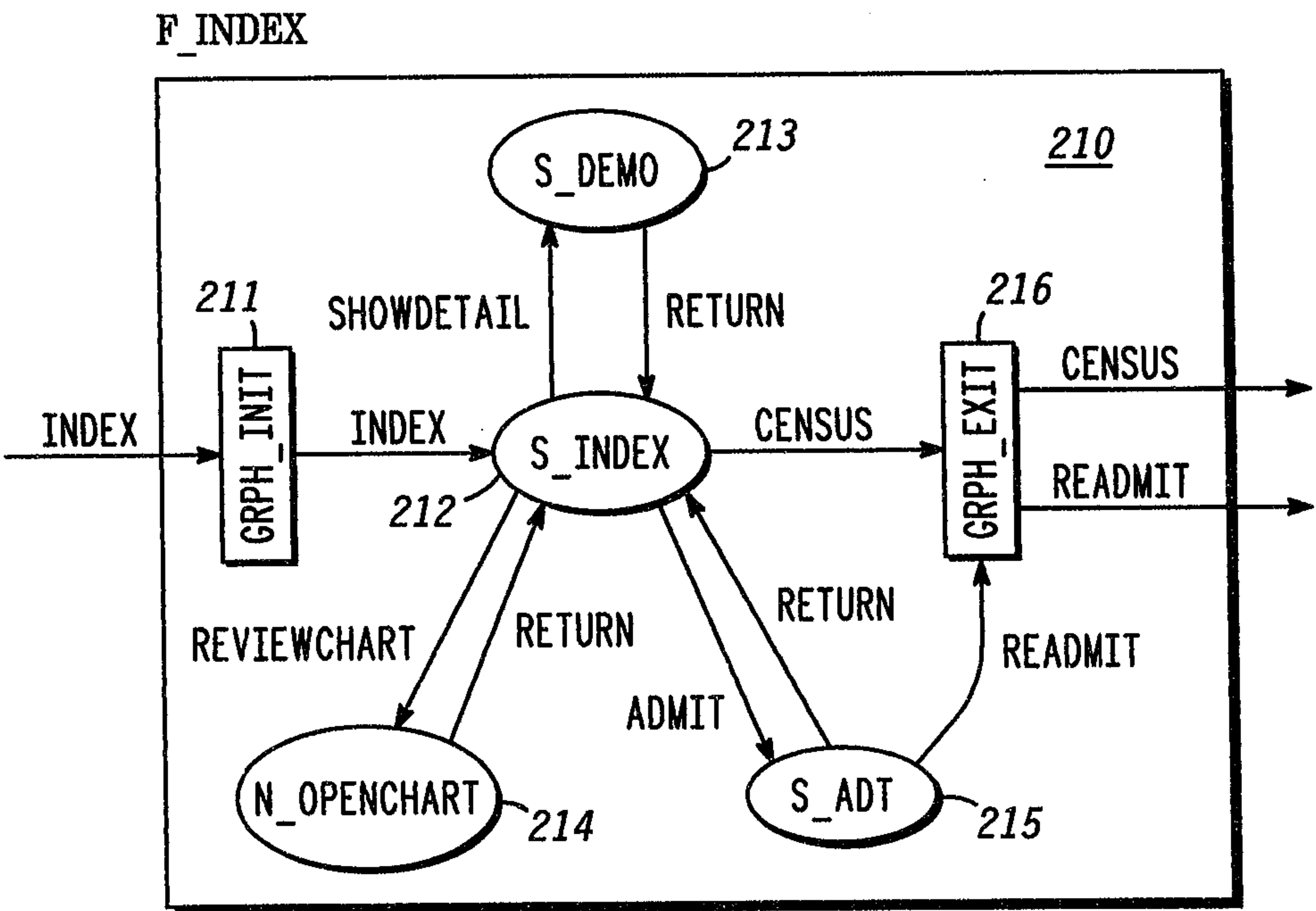


FIG. 7

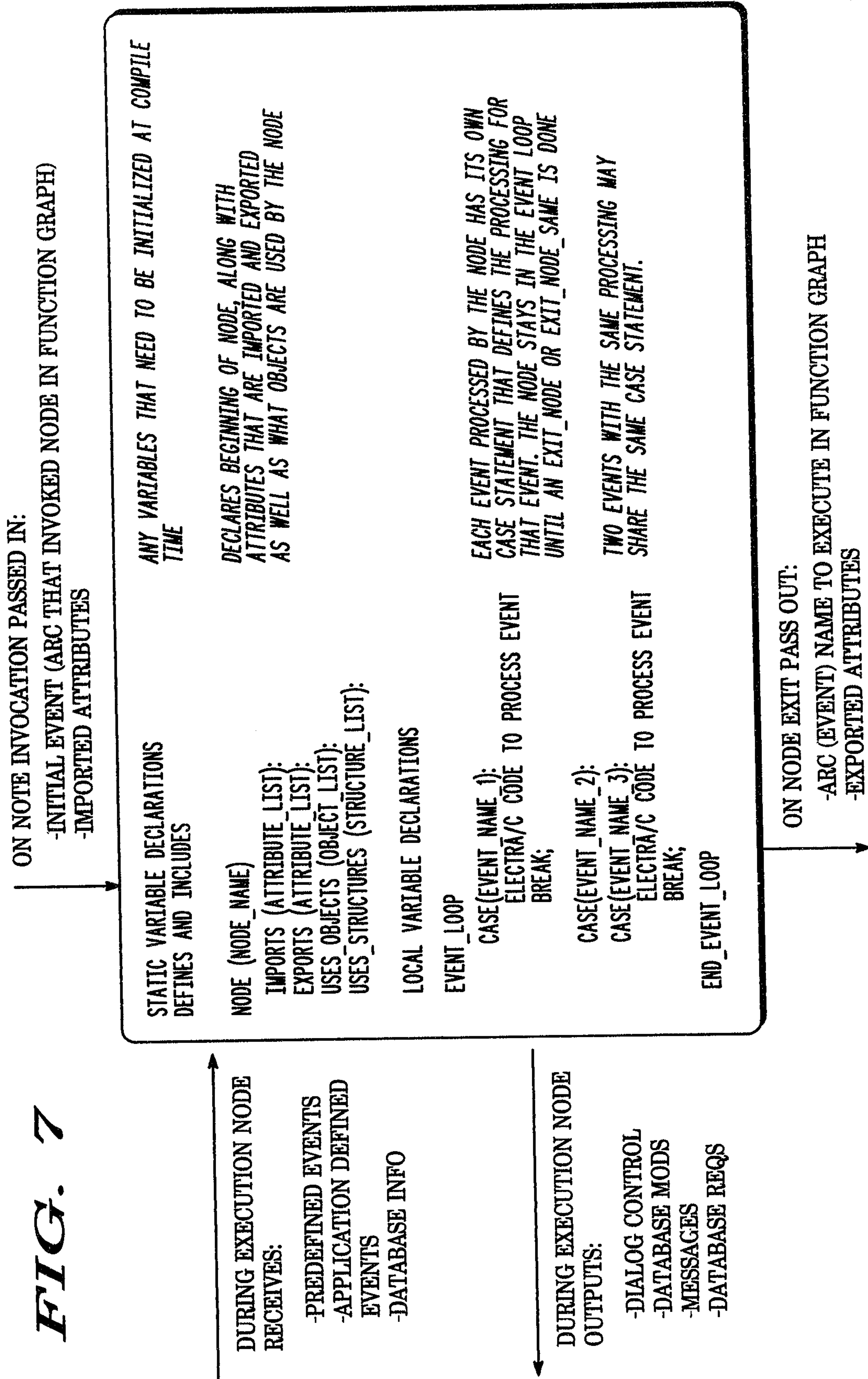


FIG. 8

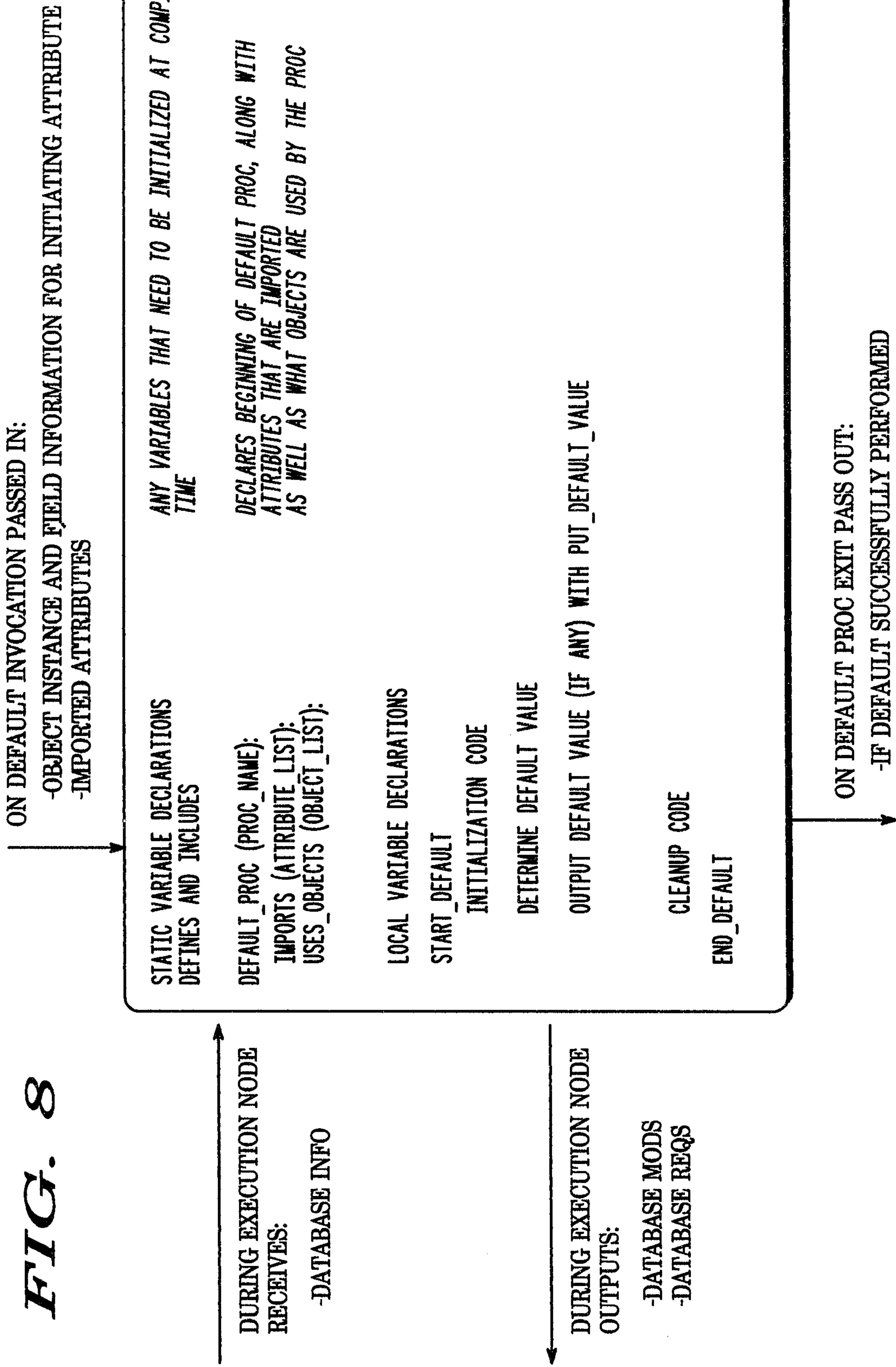
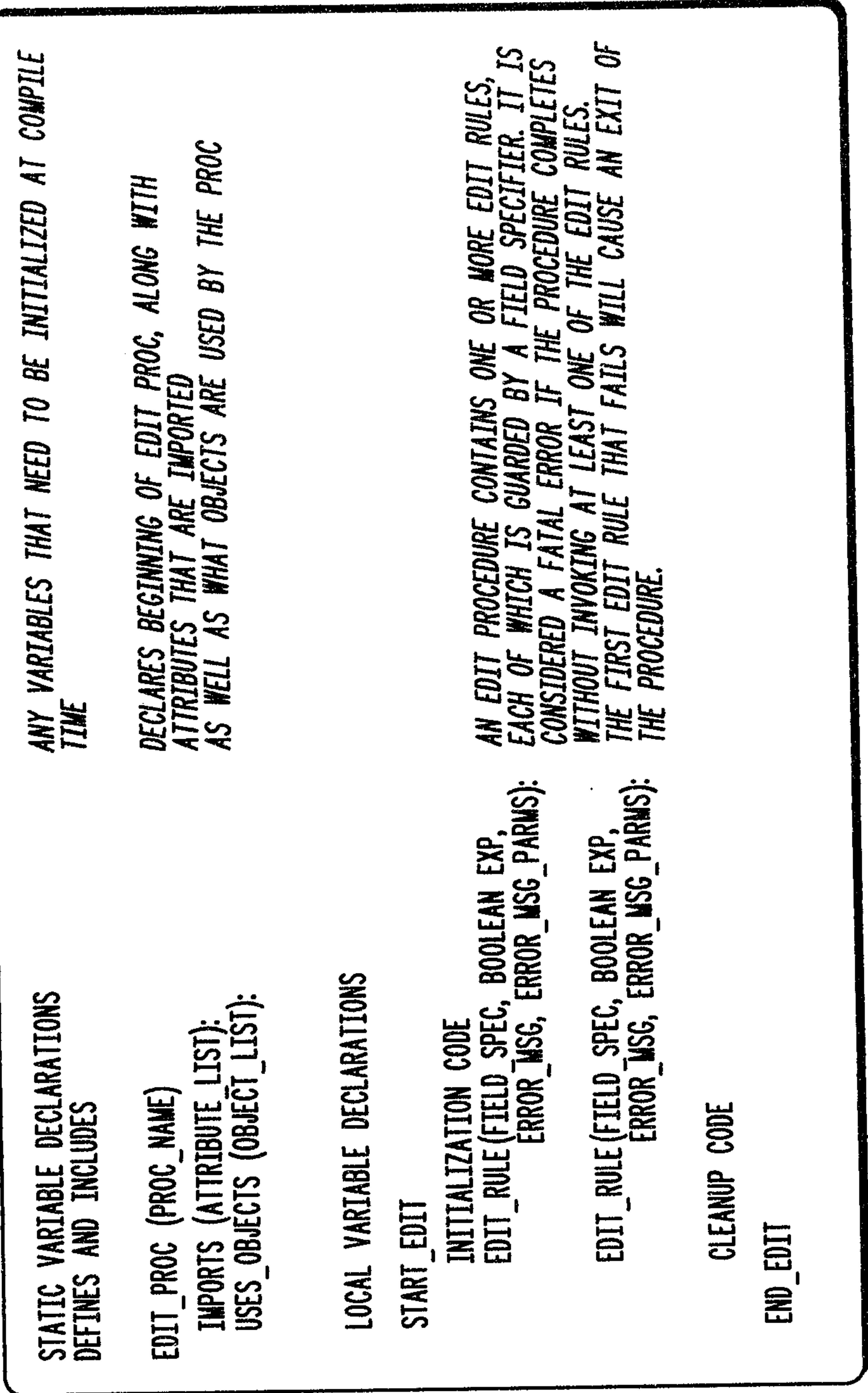


FIG. 9

ON EDIT INVOCATION PASSED IN:
-OBJECT INSTANCE AND FIELD INFORMATION FOR INITIATING ATTRIBUTE
-IMPORTED ATTRIBUTES
-WHETHER SINGLE OR CROSS FIELD EDIT



DURING EXECUTION NODE
RECEIVES:
-DATABASE INFO

DURING EXECUTION NODE
OUTPUTS:
-MESSAGES
-DATABASE REQs

ON EDIT PROC EXIT PASS OUT:
-SUCCESS OR FAILURE OF THE EDIT

FIG. 10

ON CALCULATION PROCEDURE INVOCATION PASSED IN:
-OBJECT INSTANCE AND FIELD INFORMATION FOR TRIGGER ATTRIBUTE
-IMPORTED ATTRIBUTES

STATIC VARIABLE DECLARATIONS
DEFINES AND INCLUDES

ANY VARIABLES THAT NEED TO BE INITIALIZED AT COMPILE
TIME

CALCULATION PROC (PROC_NAME):
IMPORTS (ATTRIBUTE LIST):
USES OBJECTS (OBJECT_LIST):

DECLARES BEGINNING OF CALCULATION PROC, ALONG WITH
ATTRIBUTES THAT ARE IMPORTED
AS WELL AS WHAT OBJECTS ARE USED BY THE PROC

LOCAL VARIABLE DECLARATIONS

START_CAL (TARGET OBJECT_CLASS)
INITIALIZATION CODE

USE TRIGGER_OBJECTISA TO SCREEN OUT INVALID INVOCATIONS

GET VALUES USING GET_CALC_VALUE

DO CALCULATION

OUTPUT VALUE WITH PUT_CALC_VALUE

END_CALC

DURING EXECUTION NODE
RECEIVES:

-DATABASE INFO

DURING EXECUTION NODE
OUTPUTS:

-DATABASE REQS
-DATABASE MODS

ON CALCULATION PROC EXIT PASS OUT:
-TRIGGER ANY DEPENDENT CALCULATIONS

FIG. 11

ON LIST INVOCATION PASSED IN:
-OBJECT INSTANCE AND FIELD INFORMATION FOR INITIATING ATTRIBUTE
-IMPORTED ATTRIBUTES

STATIC VARIABLE DECLARATIONS
DEFINES AND INCLUDES

ANY VARIABLES THAT NEED TO BE INITIALIZED AT COMPILE
TIME

LIST_PROC (PROC_NAME):
IMPORTS (ATTRIBUTE_LIST):
USES_OBJECTS (OBJECT_LIST):

DECLARES BEGINNING OF LIST PROC, ALONG WITH
ATTRIBUTES THAT ARE IMPORTED
AS WELL AS WHAT OBJECTS ARE USED BY THE PROC

LOCAL VARIABLE DECLARATIONS

START_LIST

INITIALIZATION CODE

OPEN LIST TABLE WITH OPEN_TABLE
USE SELECT CONSTRUCT TO RESTRICT LIST TABLE ENTITIES, MAY
ALSO USE GET LIST_VALUE TO OBTAIN DATABASE INFORMATION TO PERFORM
THIS SELECTION
OUTPUT RD TO CULLED LIST USING EXIT_LIST
CLEANUP CODE

END_LIST

DURING EXECUTION NODE
RECEIVES:
-DATABASE INFO

DURING EXECUTION NODE
OUTPUTS:
-DATABASE REQs

ON LIST PROC EXIT PASS OUT:
-RD CONTAINING CULLED LIST

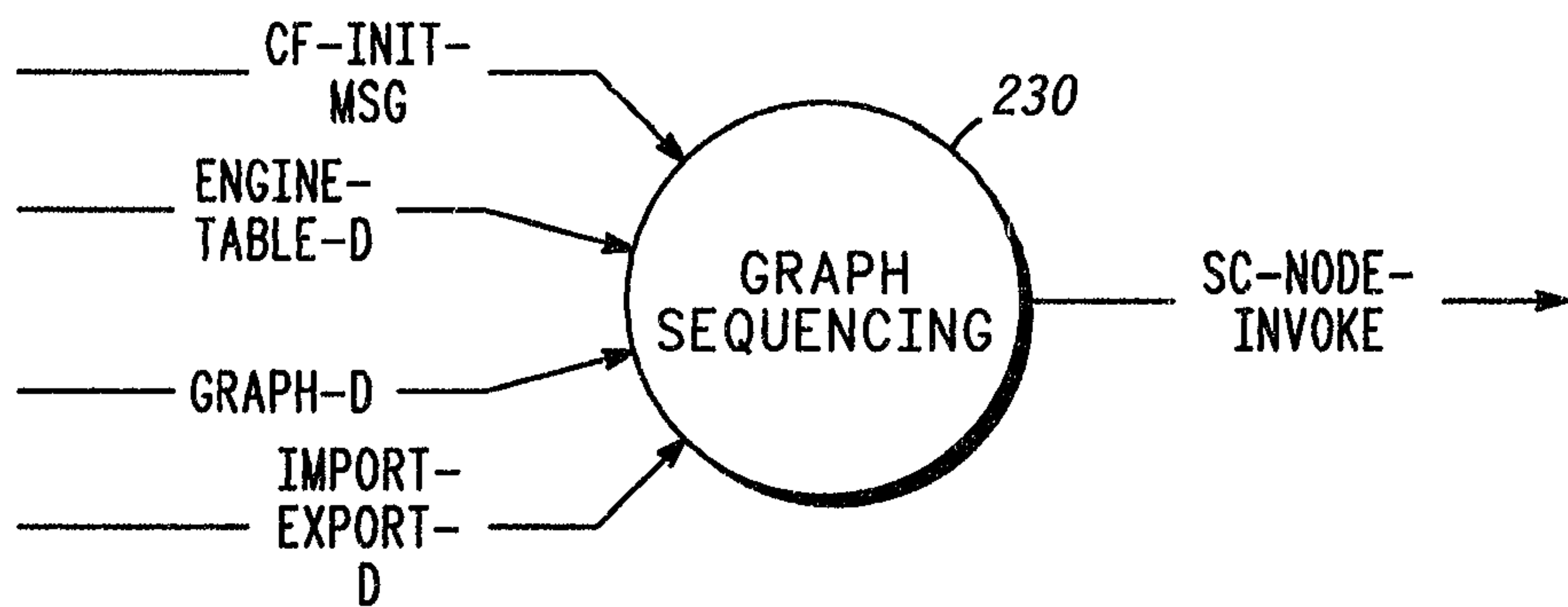
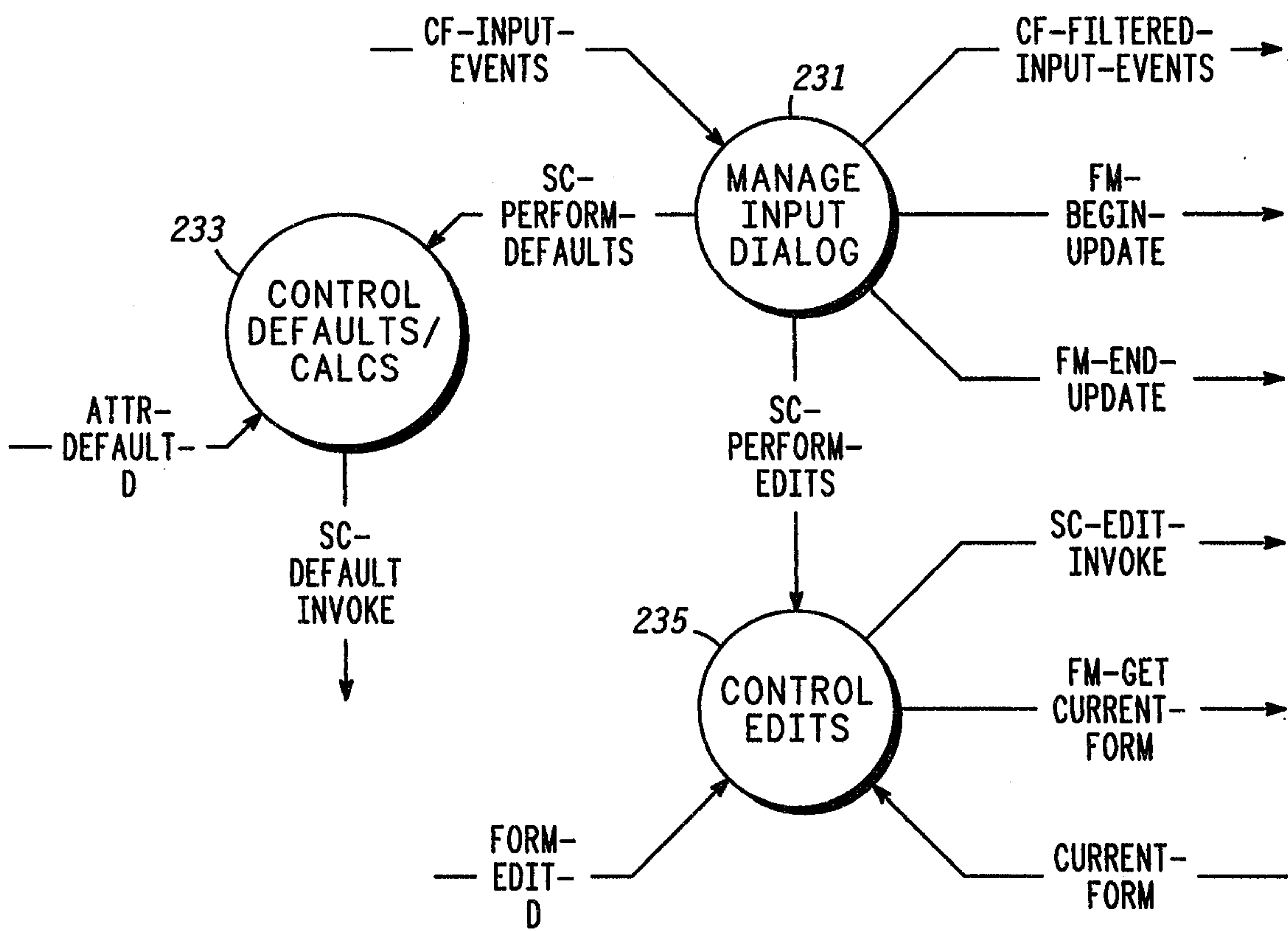


FIG. 12

FIG. 13



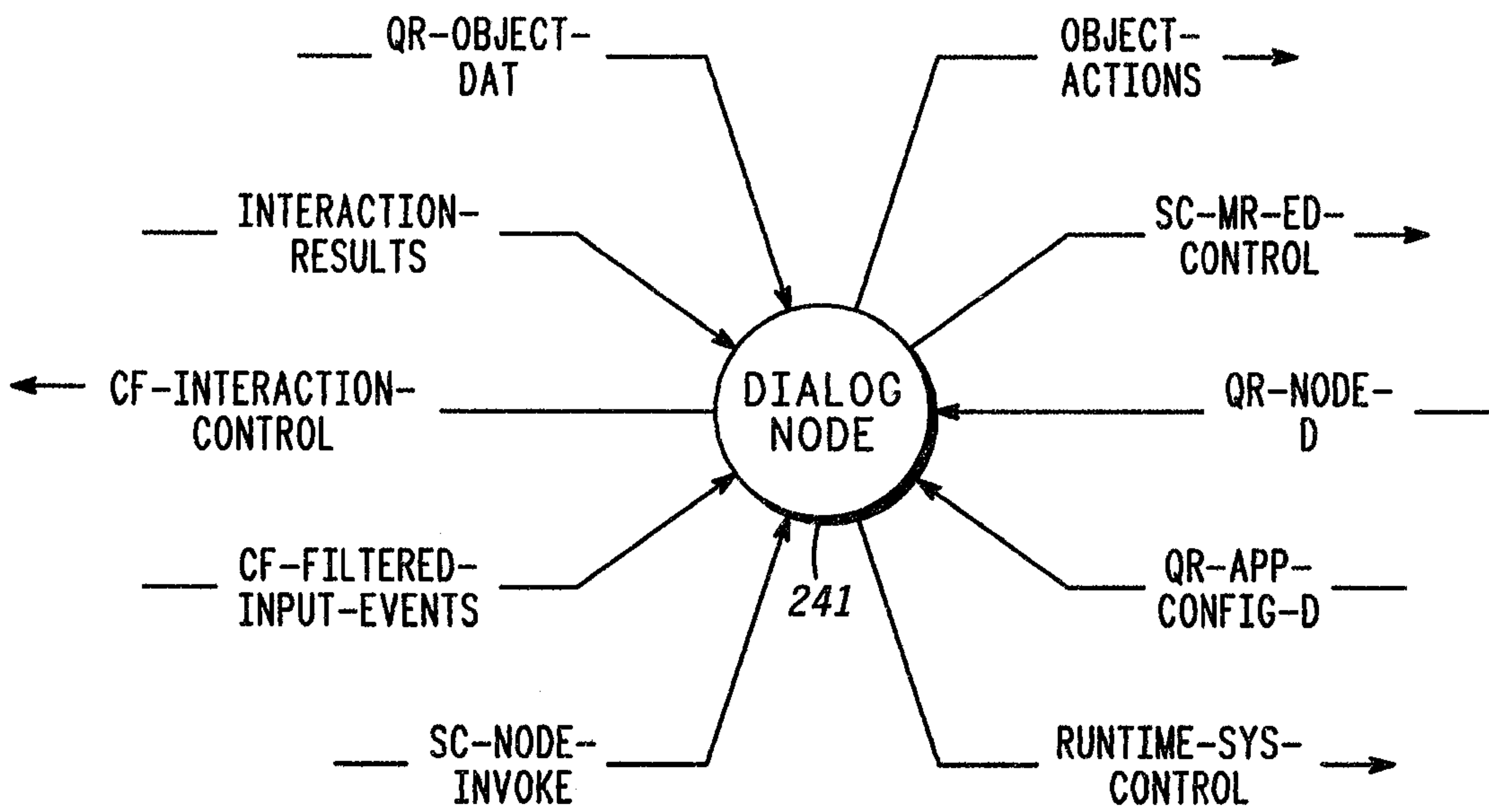
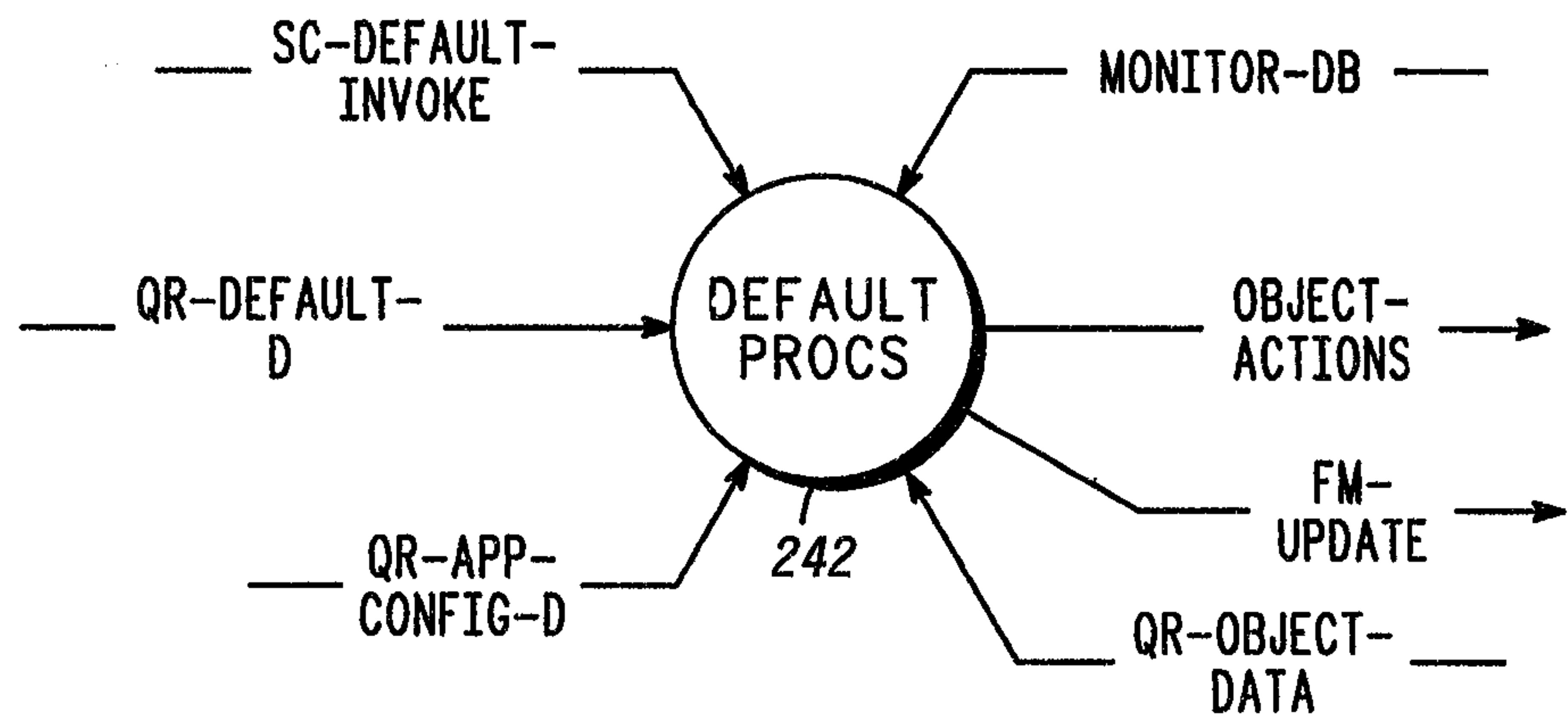
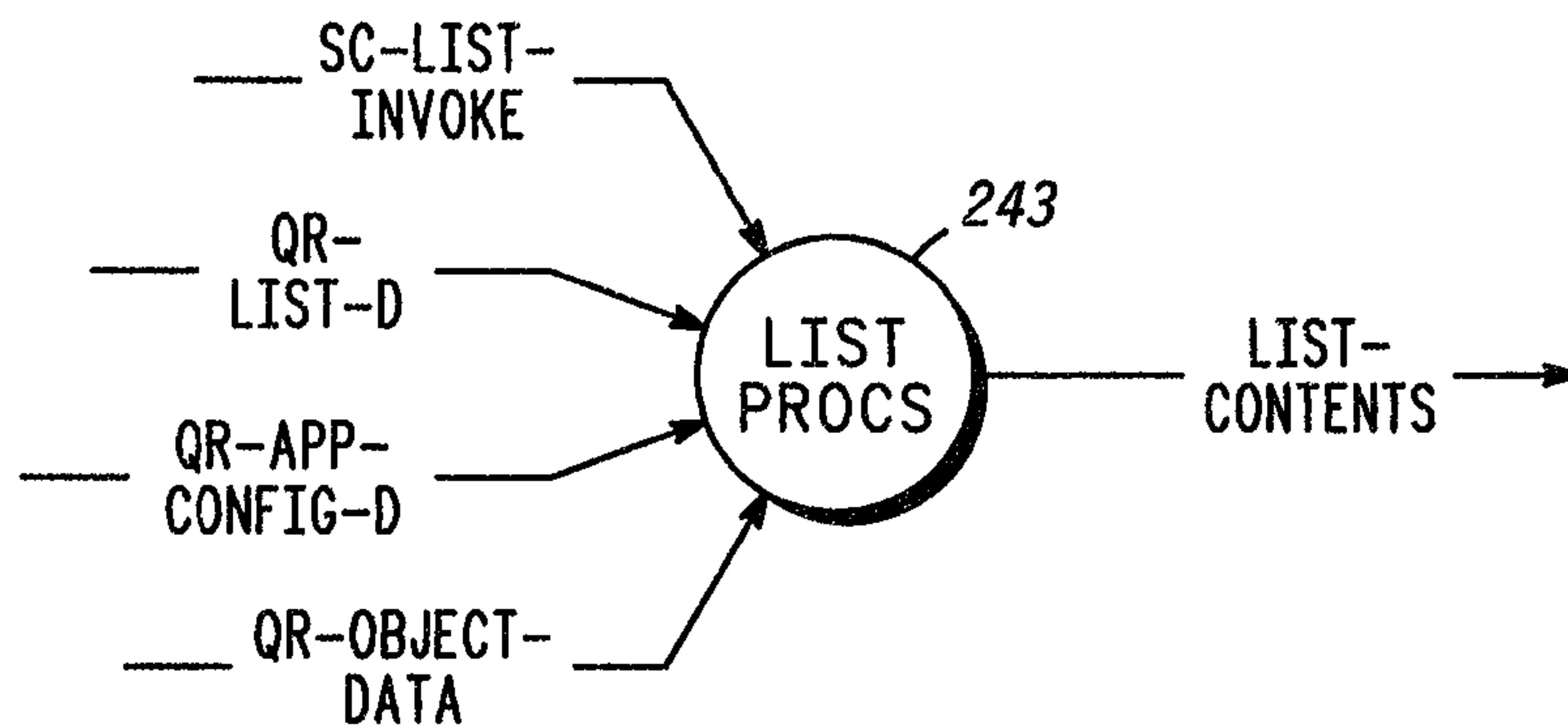
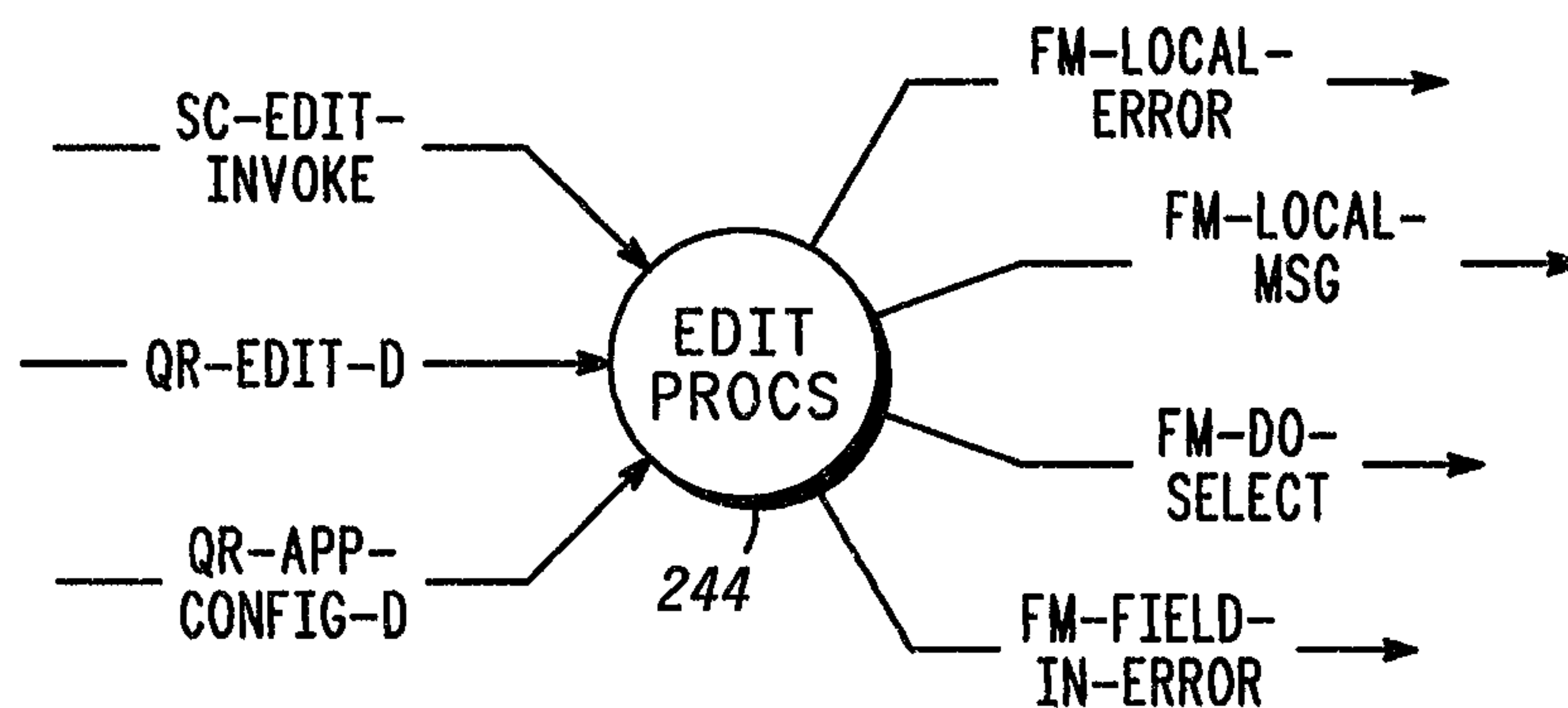


FIG. 14

FIG. 15



**FIG. 16****FIG. 17**

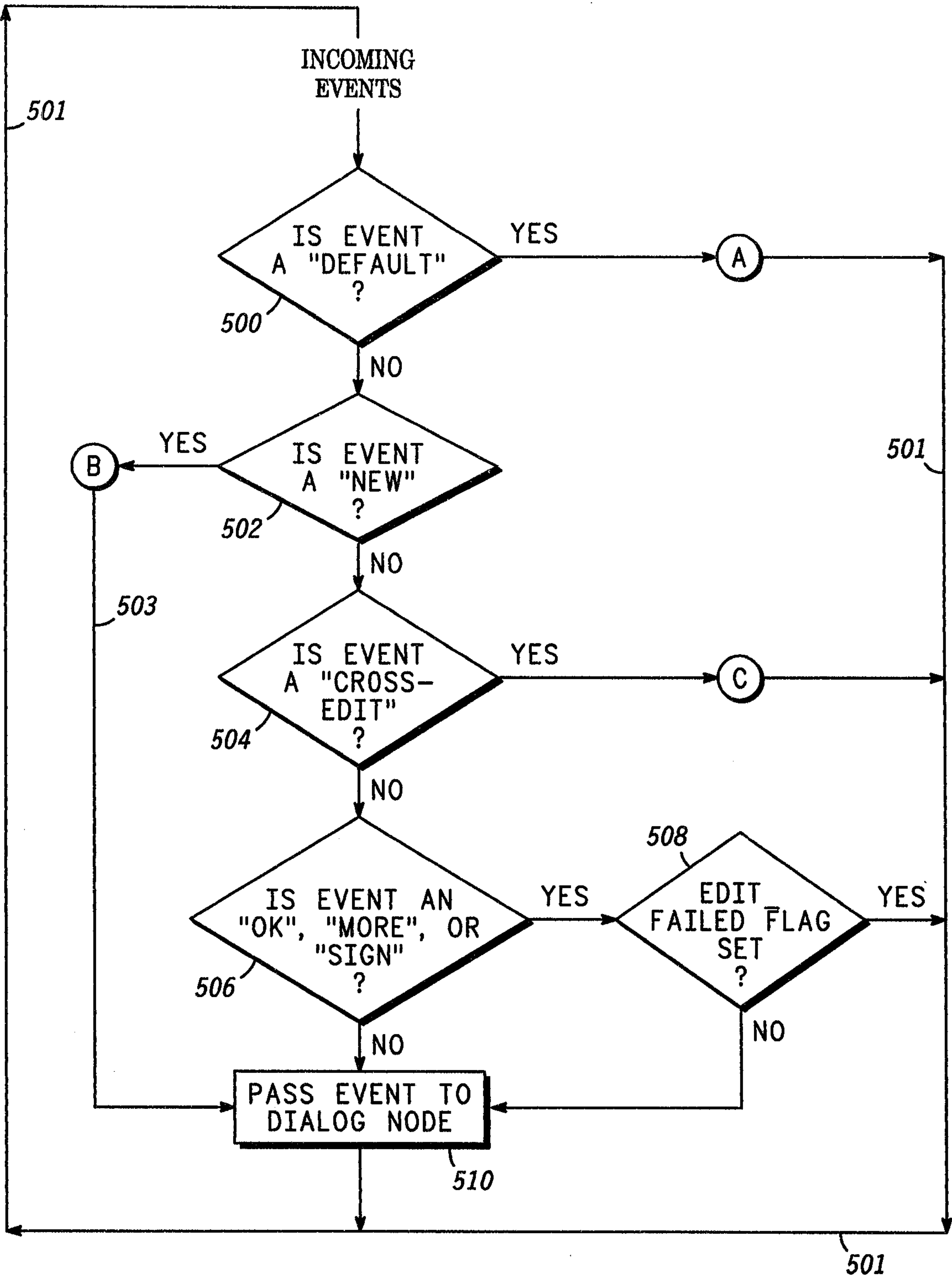
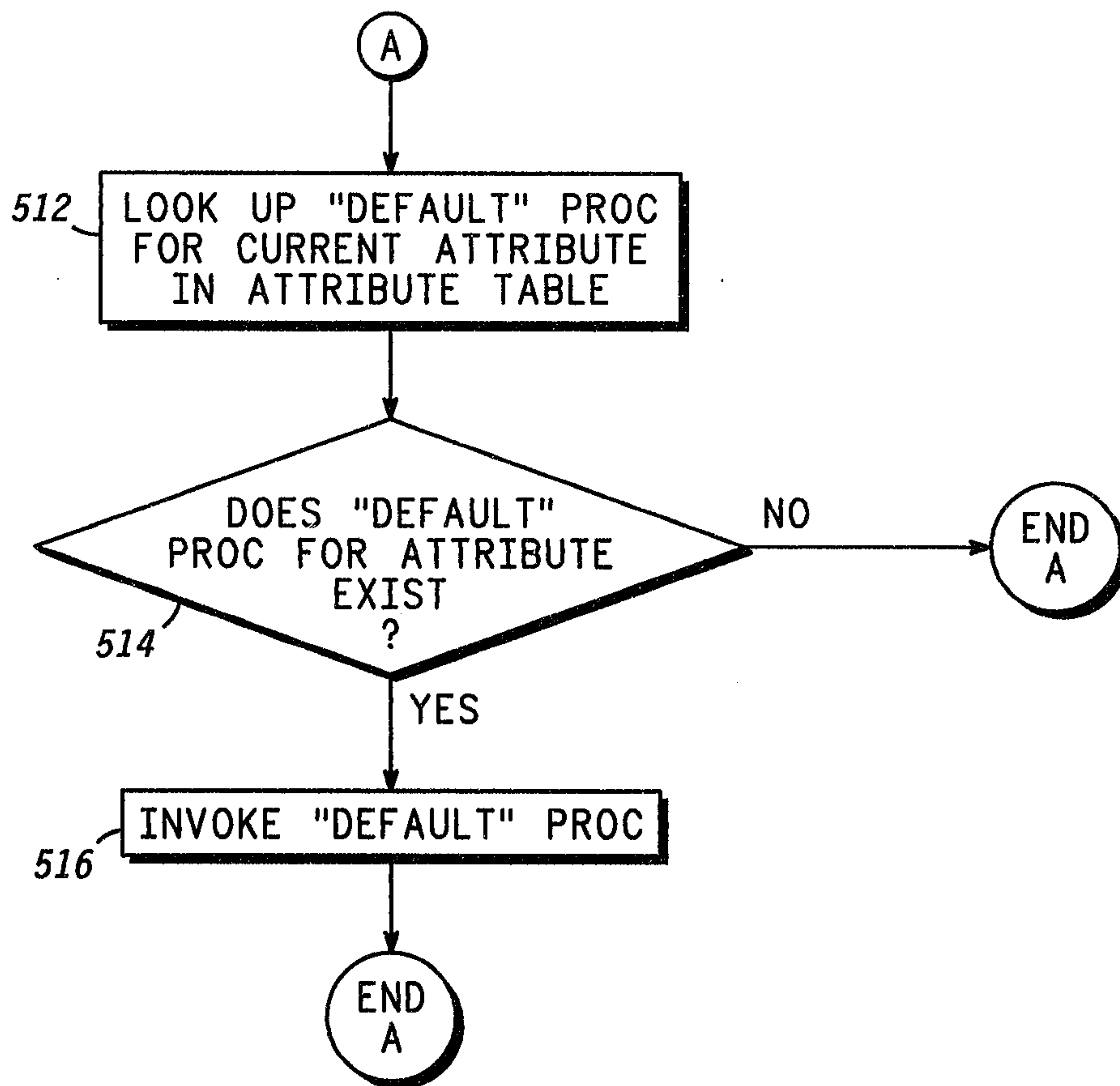
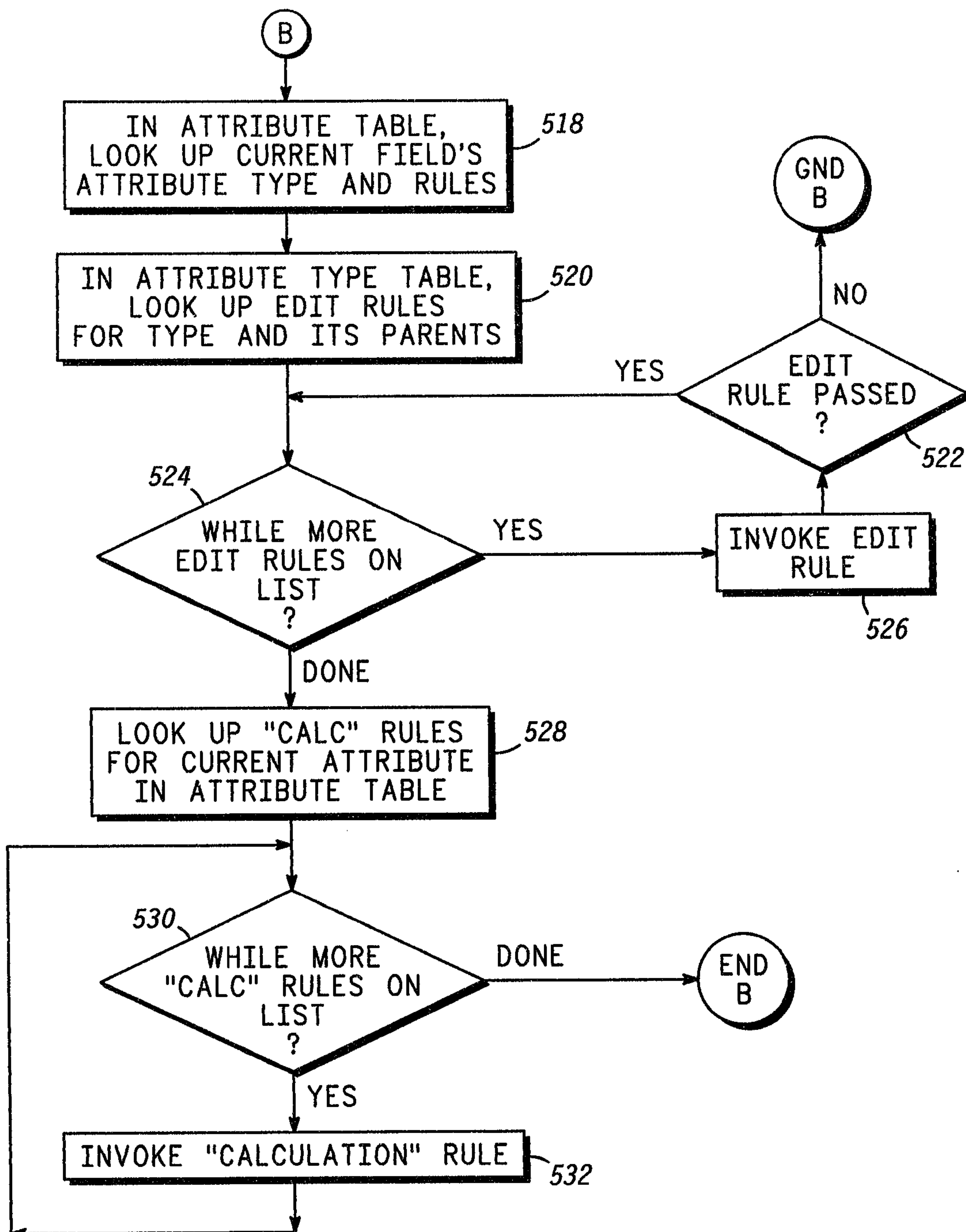
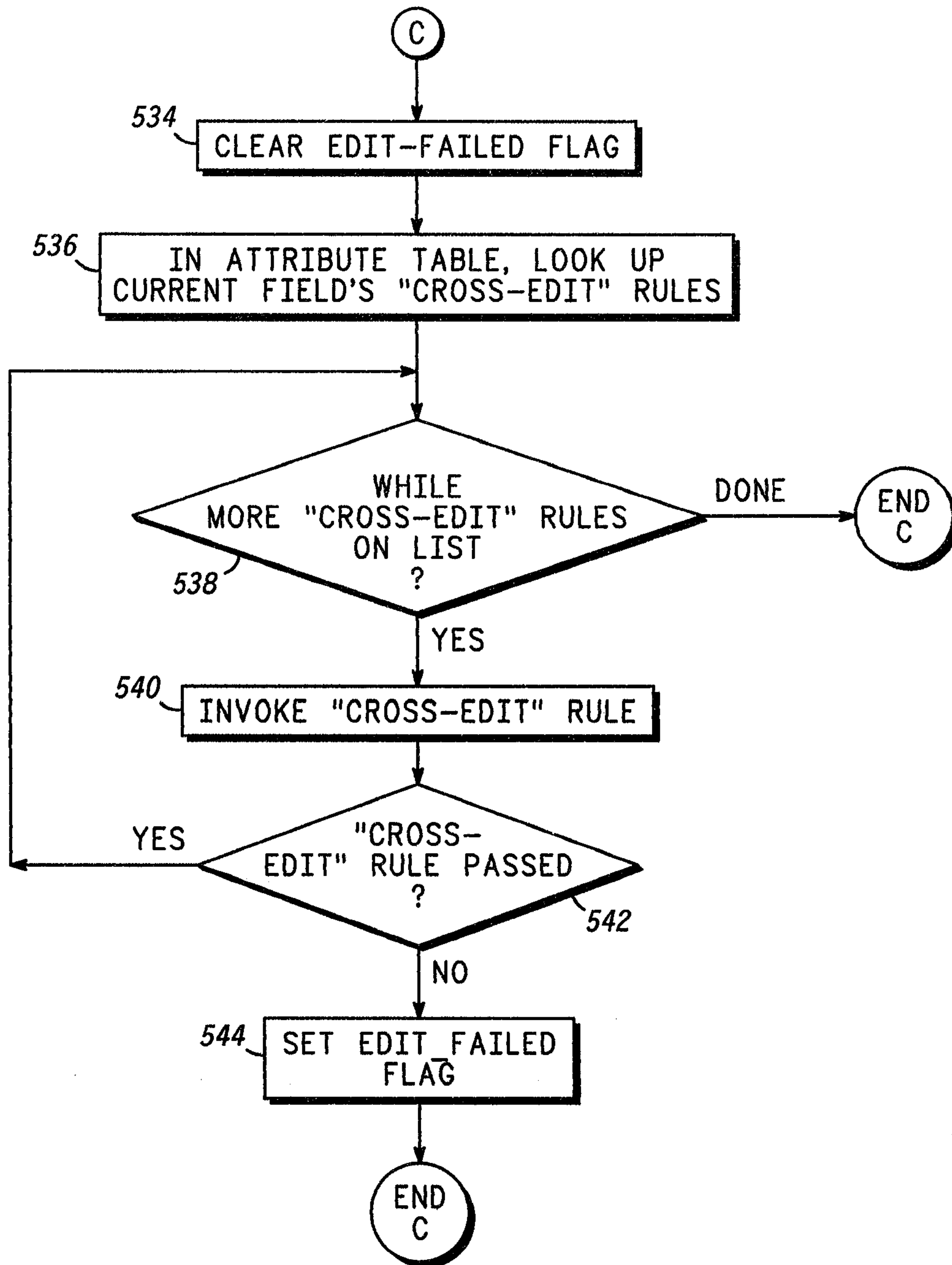


FIG. 18A

**FIG. 18B**

**FIG. 18C***Gowling, Strathy & Henderson*

**FIG. 18D**

