(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 September 2015 (03.09.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/130639 A1

(51) International Patent Classification:
G08B 25/10 (2006.01)

(21) International Application Number:
PCT/US2015/017212

(22) International Filing Date:
24 February 2015 (24.02.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/946,054 28 February 2014 (28.02.2014) US
61/973,962 2 April 2014 (02.04.2014) US
14/463,738 20 August 2014 (20.08.2014) US

(72) Inventor; and
(71) Applicant : **RASBAND, Paul, B.** [US/US]; 2981 Windswept Drive, Lantana, Florida 33462 (US).

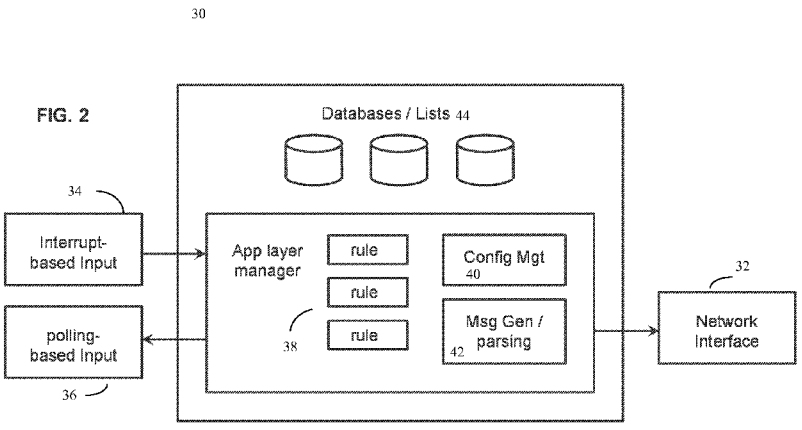
(74) Agent: **MALONEY, Denis, G.**; Fish & Richardson P.C., P.O Box 1022, Minneapolis, MN 55440-1022 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, 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, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) Title: WIRELESS SENSOR NETWORK



(57) Abstract: A networked system for managing a physical intrusion detection/alarm includes an upper tier of server devices, comprising: processor devices and memory in communication with the processor devices, a middle tier of gateway devices that are in communication with upper tier servers, and a lower level tier of devices that comprise fully functional nodes with at least some of the functional nodes including an application layer that execute routines to provide node functions, and a device to manage the lower tier of devices, the device instantiating a program manager that executes a state machine to control the application layer in each of the at least some of the functional nodes.

WO 2015/130639 A1

CLAIM OF PRIORITY

This application claims priority under 35 U.S.C. §119(e) to provisional U.S. Patent Application 61/973,962, filed on April 2, 2014, entitled: "Wireless Sensor Network", and provisional U.S. Patent Application 61/946,054, filed on February 28, 2014, entitled: "Wireless Sensor Network", and utility U.S. Patent Application 14/463,738, filed on August 20, 2014, entitled: "Wireless Sensor Network", the entire contents of which are hereby incorporated by reference.

Wireless Sensor Network

Background

This description relates to operation of sensor networks such as those used for security, intrusion and alarm systems installed on commercial or residential premises.

It is common for businesses and homeowners to have a security system for detecting alarm conditions at their premises and signaling the conditions to a monitoring station or to authorized users of the security system. Security systems often include an intrusion detection panel that is electrically or wirelessly connected to a variety of sensors. Those sensors types typically include motion detectors, cameras, and proximity sensors (used to determine whether a door or window has been opened). Typically, such systems receive a very simple signal (electrically open or closed) from one or more of these sensors to indicate that a particular condition being monitored has changed or become unsecure.

SUMMARY

However, such networks generally use a combination of wired and wireless links between the computing devices, with wireless links usually used for end-node device to hub/gateway connections. Virtually all of the devices involved in the network use some form of simple software, but in the end-nodes and hub/gateway this software is simple in form, involves little advanced capability in data reduction and decision making, and is quite static, meaning that the software typically does not change

frequently. However, when the software on these lower level devices is updated, which is not frequently, traditional boot-loading methods are used. However, these boot-loading methods are time-consuming, energy-consuming, and require rebooting of the updated device, which can present a security/alarm issue.

According to an aspect a networked sensor system includes an upper tier of server devices, the server devices including processor devices and memory in communication with the processor devices. The system also includes a middle tier of gateway devices that are in communication with one or more of the upper tier server devices and a lower level tier of devices that include fully functional sensor nodes with at least some of the fully functional sensor nodes including an application layer that executes routines to provide node sensor functions and an application layer manager to manage the application layer in the at least some of the functional nodes in the lower tier of devices.

Aspects can also include methods, computer program products and systems.

One or more advantages may be provided from one or more of the above aspects.

The network can use a combination of wired and wireless links, preferable wired between the tiers, especially with wireless links between the middle and lower tier connections (for example, end-node device to hub/gateway). The devices involved in the network can include advanced capabilities areas such as data reduction and decision making, and the capabilities of the device are dynamically changeable, meaning that the software can be updated without the traditional boot-loading methods avoiding the time-consuming, energy-consuming, and rebooting required by the updated device, thus avoiding potential security/alarm issues when such sensor and other end node devices are updated. This enables management of such sensors and other end node devices having advanced capabilities in data reduction and decision making.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention is apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a schematic diagram of an exemplary networked security system.

FIG. 2 is a block diagram of generic application layer manager.

FIG. 3 is a block diagram showing an example process on the application layer manager.

FIG. 4 is a diagram of exemplary state transitions on the application layer manager.

FIGS. 5 and 6 are flow charts.

FIG. 7 is a block diagram of components of an example networked security system.

DETAILED DESCRIPTION

Described herein are examples of network features that may be used in various contexts including, but not limited to, security/intrusion and alarm systems. Example security systems may include an intrusion detection panel that is electrically or wirelessly connected to a variety of sensors. Those sensors types may include motion detectors, cameras, and proximity sensors (used, e.g., to determine whether a door or window has been opened as well as other types of sensors). Typically, such systems receive a relatively simple signal (electrically open or closed) from one or more of these sensors to indicate that a particular condition being monitored has changed or become unsecure.

For example, typical intrusion systems can be set-up to monitor entry doors in a building. When a door is secured, a proximity sensor senses a magnetic contact and produces an electrically closed circuit. When the door is opened, the proximity sensor opens the circuit, and sends a signal to the panel indicating that an alarm condition has occurred (e.g., an opened entry door).

Data collection systems are becoming more common in some applications, such as home safety monitoring. Data collection systems employ wireless sensor networks and wireless devices, and may include remote server-based monitoring and report generation. As described in more detail below, wireless sensor networks generally use a combination of wired and wireless links between computing devices, with wireless

links usually used for the lowest level connections (e.g., end-node device to hub/gateway). In an example network, the edge (wirelessly-connected) tier of the network is comprised of resource-constrained devices with specific functions. These devices may have a small-to-moderate amount of processing power and memory, and may be battery powered, thus requiring that they conserve energy by spending much of their time in sleep mode. A typical model is one where the edge devices generally form a single wireless network in which each end-node communicates directly with its parent node in a hub-and-spoke-style architecture. The parent node may be, e.g., an access point on a gateway or a sub-coordinator which is, in turn, connected to the access point or another sub-coordinator.

Referring now to FIG. 1, an exemplary (global) distributed network 10 topology for a Wireless Sensor Network (WSN) is shown. In FIG. 1 the distributed network 10 is logically divided into a set of tiers or hierarchical levels 12a-12c.

In an upper tier or hierarchical level 12a of the network are disposed servers and/or virtual servers 14 running a "cloud computing" paradigm that are networked together using well-established networking technology such as Internet protocols or which can be private networks that use none or part of the Internet. Applications that run on those servers 14 communicate using various protocols such as for Web Internet networks XML/SOAP, RESTful web service, and other application layer technologies such as HTTP and ATOM. The distributed network 10 has direct links between devices (nodes) as shown and discussed below.

The distributed network 10 includes a second logically divided tier or hierarchical level 12b, referred to here as a middle tier that involves gateways 16 located at central, convenient places inside individual buildings and structures. These gateways 16 communicate with servers 14 in the upper tier whether the servers are stand-alone dedicated servers and/or cloud based servers running cloud applications using web programming techniques. The middle tier gateways 16 are also shown with both local area network 17a (e.g., Ethernet or 802.11) and cellular network interfaces 17b.

The distributed network topology also includes a lower tier (edge layer) 12c set of devices that involve fully-functional sensor nodes 18 (e.g., sensor nodes that include

wireless devices, e.g., transceivers or in some implementations just transmitters or receivers, which in FIG. 1 are marked in with an "F") as well as constrained wireless sensor nodes or sensor end-nodes 20 (marked in the FIG. 1 with "C"). In some embodiments wired sensors (not shown) can be included in aspects of the distributed network 10.

Constrained computing devices 20 as used herein are devices with substantially less persistent and volatile memory compared to other computing devices, sensors in a detection system. Currently examples of constrained devices would be those with less than about a megabyte of flash/persistent memory, and less than 10-20 kbytes of RAM/volatile memory). These constrained devices 20 are configured in this manner; generally due to cost/physical configuration considerations.

In a typical network, the edge (wirelessly-connected) tier of the network is comprised of highly resource-constrained devices with specific functions. These devices have a small-to-moderate amount of processing power and memory, and often are battery powered, thus requiring that they conserve energy by spending much of their time in sleep mode. A typical model is one where the edge devices generally form a single wireless network in which each end-node communicates directly with its parent node in a hub-and-spoke-style architecture. The parent node may be, e.g., an access point on a gateway or a sub-coordinator which is, in turn, connected to the access point or another sub-coordinator.

Each gateway is equipped with an access point (fully functional node or "F" node) that is physically attached to that access point and that provides a wireless connection point to other nodes in the wireless network. The links (illustrated by lines not numbered) shown in FIG. 1 represent direct (single-hop network layer) connections between devices. A formal networking layer (that functions in each of the three tiers shown in FIG. 1) uses a series of these direct links together with routing information used at intermediate routing-capable devices to send messages (fragmented or non-fragmented) from one device to another over the network.

The WSN 10 implements a state machine approach to an application layer that runs on the lower tier devices 18 and 20. Discussed below is an example of a particular implementation of such an approach. States in the state machine are comprised of sets

of functions that execute in coordination, and these functions can be individually deleted or substituted or added to in order to alter the states in the state machine of a particular lower tier device.

The WSN state function based application layer uses an edge device operating system (not shown, but such as disclosed in the above mentioned provisional application) that allows for loading and execution of individual functions (after the booting of the device) without rebooting the device (so-called "dynamic programming"). In other implementations, edge devices could use other operating systems provided such systems allow for loading and execution of individual functions (after the booting of the device) preferable without rebooting of the edge devices.

Referring now to FIG. 2, an embodiment an application layer manager 30 is shown. The application layer manager 30 is generic in the sense that the application layer manager 30 does not depend upon a specific application solution or "business" logic details in the devices that are updated, e.g., devices 18, 20 (lower tier 12c FIG. 1). The application layer manager 30 handles "pass-offs" (changes in functions currently operating) from function to function, on, e.g., the nodes, e.g., devices 18, 20. These pass-offs are requested by actual state functions executing in the nodes or devices 18, 20.

The application layer manager 30 accomplishes such "pass-offs" (changes in functions currently operating) using a transition table (FIG. 4) that serves as a central descriptor for the state functions. Inputs to the application layer manager 30 include parsed messages from the network layer via interface 32. The application layer manager 30 includes interrupt and polling based inputs via processor peripheral interrupts from interrupt interface 34 and polled sensor/peripheral inputs via interface 36.

The application layer manager 30 involves characterizing inputs sufficiently to apply rules 38 that dictate changes in configuration, stored data, and/or precipitate message generation. The application layer manager 30 has rules 38 and a configuration manager 40 as well as a message generator/parser 42. The application layer manager 30 uses network message and sensor / processor peripheral based inputs, local data

stores 44 (for transition table) and lists, configuration management functions, rules set, and report generation capabilities as shown.

Edge Application Layer Architecture

Referring to FIG. 3, an application module set 50 includes an application layer 50 for the edge devices 18 and 20 (FIG. 1) is shown. The application module set 50 includes a layer 52 that is managed by the application layer manager (FIG. 4) and a layer 54 that is not managed by the application layer manager. In this embodiment, the application layer manager is separate from, e.g., isolated from these other firmware modules used at the edge of the WSN (e.g., wireless web modules, EDFF's, etc. not shown) in order to apply changes in the application layer code without requiring changes to code in these other modules. In addition, as shown real-time processing of motion ISR and motion filter are not handled by the application layer manager, whereas motion report generator and heart beat generator are handled by the application layer manager.

The application module set 50 depicted in the example of FIG. 3 includes functions managed by the application layer, e.g., a motion report generator 53a and heartbeat generator 53b that are in communication with a stack 56. Changes to the application layer 52 are possible by having an understanding of the details of the workings of the application layer 52 without the need to fully understand all of the details of these other isolated modules. This is desirable as different groups of individuals and/or systems may be tasked with the coding and maintenance of the respective modules. Also, the application layer 52 is configured in a general way that supports the upgrading of portions of the application layer (e.g., individual business rules, reports, filters, and other functions) without requiring updating of the entire application layer.

FIG. 3 in addition, depicts a state diagram among various processes running in the application module set and with interaction with a transition table as set out below.

Function Name	Function ID	Allowed Transitions	Function Index	Execution Type

(assigned externally)	**(assigned externally)**	**(assigned externally)**	**(assigned by AppMgr)**	**(assigned externally)**

Referring now to FIG. 4, an exemplary situation involving the Application layer manager 50 (App_Mgr()) where there are two states (State 1 with functions A, B, and C, and State 2 with functions D and E) is shown. The transition table governs state transitions.

The transition table shows what state (or states in the case of a nondeterministic finite automaton) a finite semi-automaton or finite state machine will move to, based on the current state of the machine and other inputs. A state table is essentially a truth table in which some of the inputs are the current state and the outputs include the next state along with other outputs. A state table is one of several ways to specify a state machine, other ways being a state diagram, and a characteristic equation.

State 1 is the normal state and has an entry point, "Func A." Normally, State 1 executes "Func A" which requests "Func B" which requests execution of "Func C." In the example, a condition occurs (actual condition is implementation specific and the detail of which is not necessary to understand what follows). Under this situation with the condition occurring State 1 transitions to State 2 when "Func B" requests execution of "Func D" rather than "Func C." State 2 may exists for only one cycle (D-E-exit) or many cycles (D-E-D-E-…-exit). However, when the exit occurs in this example, it does so without calling any function. By default then, the AppMgr's Idle function runs Func A since it is the entry point function.

App Layer Modularity

Referring to FIG. 5, a "typical" application 60 on a motion sensor handles a motion sensor's raw data with an interrupt routine, the motion sensor interrupt service routing directly calls a motion filter that maintains its own state and declares, when appropriate, a "motion event" 62. The motion event is handled 64 (after perhaps a bit of un-deterministic latency) by a motion report generator that calls a wireless program stack to place the report into that stack's out-going queue. The motion report generator

waits 66 for an ACK message and re-sends the message, as necessary, until an ACK is received. A heartbeat message is generated 68 periodically and placed into the wireless stack out-going message queue, and an ACK is awaited. (Heartbeat messages are not re-submitted after delivery failure, but a new heartbeat message is not sent until the result of the previous send is obtained from the wireless stack.)

Referring now to FIG. 6, the application layer is configured to satisfy the requirement of modularity by defining and linking together different portions of the application layer so that individual portions are updatable over the wireless link without breaking the overall application. The application layer instantiates 72 an "object" that is a fundamental building block for application layer "machine." The object has an array of function pointers, with each function serving as a "keeper", e.g., holding of a particular state and a special manager function (application layer manager 30 or "AppMgr") that tracks 74 which function is running in the machine (i.e., which array index is in effect).

State transitions are accomplished by the current function transferring 76 function control to the next appropriate function marking entrance into the new states (e.g., FuncD in FIG. 4) by changing 76 this index (or asking AppMgr to change the index. The AppMgr is general with "hardwired" business logic residing in the individual state functions, rather than in the AppMgr and individual states are changed by replacing the corresponding function with a new version of that function, transmitted from an external host such as a gateway. Thus, making changes to an allowed state transition (either adding a new transition or deleting an old one) is accomplished by replacing the functions which participate in the state change with the new functions.

In some embodiments, the AppMgr is configured 78 to actually change the current index value for the current state function, whereas in others the old function directly activates 80 the new function, without using the AppMgr() as an intermediary. This is because the AppMgr has a mapping 82 of allowed transitions and check 84 for violations of this mapping (i.e., a given function tries to pass control to another function and in so doing make a state transition that is not allowed). This helps to verify that changes to the state machine behavior are valid and that changes to the state machine

behavior actually take place, since an error message will be generated by AppMgr() 86 when an erroneous state change is requested by a function. Otherwise, individual states are changed by replacing the corresponding function with a new version of that function, 88.

Example Application

Let p_AppFunc[i] be a pointer to the i^{th} application function. Let N_i be the "current index" value maintained by AppMgr(). N_i is a global variable that retains its value from one pass through AppMgr() to the next.

AppMgr is a root function that is executed by a (EDFF) scheduler such as in the operating system running on the edge device. AppMgr runs completely through every few milliseconds. Each time AppMgr() runs, AppMgr executes the function pointed to by p_AppFunc[N_i]. In some embodiments, the state machine can be implemented as a set of arrays, whereas in more complex implementations the state machine is implemented as a set of functions that are linked through a linked list to allow for an indeterminate number of states in the state machine.

For some states, only one function call may be required. That is, p_AppFunc[N_i] would run once and then N_i would change to a different value, say N_k so that on the next call of AppMgr(), a different state would be entered (i.e., p_AppFunc[N_k] would run). For other states the corresponding function might run many times before N_i changes. An example of the single-run function would be the sending of a report. An example of the multi-run function would be the activity of a sensor filter that acts on raw data from a sensor device.

The various functions p_AppFunc[i] not only decide when they should request that AppMgr() make a state change, but these functions indicate what new function(s) (e.g., what new value(s) of N_i) AppMgr() should choose from, as AppMgr() is configured to be fairly generic and thus all business logic, including the description of transitions between states, is contained in the p_AppFunc[] functions.

Simultaneous actions

The two p_AppFunc[] functions need to have different tasks done at the same time, for example, simultaneously filtering data from two sensors (e.g., de-bouncing a

switch and filtering motion data from an accelerometer. One general approach to providing an AppMgr is to run two state functions at a time (execute both with each pass through AppMgr). Another approach keeps AppMgr simple and just requires that application states provided with code to make calls to each other and thus invoke each other. That is, p_AppFunc[N_i] requests execution of p_AppFunc[N_k] upon termination of execution of p_AppFunc[N_i] and vice versa. In effect, the two app functions split the time and attention of AppMgr without AppMgr planning for time sharing.

The various versions of p_AppFunc[] are maintained in an executable code repository in the gateway and cloud, and each such function can have an ID number that is used to differentiate one version of a function from another (and in many cases the differences in the generations or versions of the function may be small but important, so getting exactly the right ID number). Inside a given function, the requested change to a different function or state becomes very specific with respect to a code version, therefore there is logical that the parameter used by a function to request a state change (function change) is actually the ID number of the new function.

A simple way to manage versions is to give App Layer state functions their own file type. File type is maintained as one of the fields in the file index in, e.g., flash memory so that the initialization process of AppMgr() during bootup searches for files of that type in the flash memory, and produces the array of function pointers, with index i running from 0 to a maximum value, i_max.

During this initialization AppMgr() maps each value i to a function, p_AppFunc[i] and that function's corresponding function ID, and produces a table showing, for each function ID, the corresponding index value i, and the allowed state transitions (function IDs for functions reachable from the current function).

For example, during the course of operation of the current function, e.g., p_AppFunc[N_i] with its function ID,e.g., 0x31C7, the function might return to AppMgr() a return value of 0x396B. This return value is a request to run the function p_AppFunc[] having the function ID "0x396B." AppMgr() uses the state table to determine if request to run p_AppFunc[] having the function ID "0x396B is a transition that is allowed for function ID 0x31C7, and if so, what value of i corresponds to

function 0x396B. If it is a valid request, AppMgr() sets N_i equal to the new value of i corresponding to function ID "0x396B and, upon next execution of AppMgr(), the new function ID "0x396B would run.

During initialization of AppMgr() and the producing of the state table, simple graph analysis algorithms run to ensure that each state is reachable (no states are isolated) and to make sure that all states are part of a single state machine (i.e., there are not two smaller and totally separated sets of states). The state table validation also requires that no state transition can involve a transition to a non-existent function.

AppMgr() always has a stem state (idle state) with function ID 0x0001 that runs when the index N_i is undefined. Initialization of the state machine is done in a second state (initialize state) with its own generic function, with function ID 0x0000. Any time any of the functions change (e.g., by a wireless download of a new function or functions), AppMgr() will re-run function 0x0000 and then transition to state function 0x0001. It is a further requirement that one and only one of the user-supplied functions in the complete function set is identified as the entry state. This is the state called by 0x0001 automatically. From that point on the user-supplied functions request the state changes as part of their return values.

Returning to FIG. 3 the state diagram and the table mentioned above, the table below is now populated with the state transitions for the example application described above.

Function Name (assigned externally)	**Function ID** (assigned externally)	**Allowed Transitions** (assigned externally)	**Function Index** (assigned by AppMgr)	**Execution Type** (assigned externally)
Motion ISR	0x31C7	0x31A2	2	On event
Motion Filter	0x31A2	0x31C7 0x3362	3	On event
Motion Report Generator	0x3362	None (AppMgr Idle)	4	On call

Heartbeat Generator	0x33EB	None (AppMgr Idle)	5	Perpetual (this is also entry function)
AppMgr Idle	0x0001		1	Perpetual
AppMgr Initialize	0x0000		0	On boot, or when called by AppMgr Idle

The above example is simple for purposes of illustration of the concept. However, more complex application sets can also be used. For example, suppose there are two "states" in a node – the first state corresponding to the perpetual running of functions A, B, and C in a continuous loop and the second state corresponding to the running of functions D and E in another perpetual loop. In the first state (the normal state) function A (the "entry function") runs to completion and requests to AppMgr() that it run function B. When function B is complete, it requests function C, which in turn requests function A. Because function A is the entry function and the loop A-B-C-A … is a closed loop, functions D and E will normally not run. However, under special conditions suppose function B, when it exits, requests function D rather than C. Function D and E then run in a loop (D-E-D-E…) until one of them requests a function in the first loop. In this way, functions or sets of functions correspond to states, and functions operating in each state manage the state transitions.

When a function exits with no new function requested, AppMgr Idle will simply run the entry point function again. In some cases with very simple nodes, there may be NO entry function, in which case the idle function will just run itself until an event-initiated function is run.

Referring back to FIG. 4, a hypothetical (generic) situation where there are two states (State 1 with functions A, B, and C, and State 2 with functions D and E). State 1 is the normal state and has the entry point, Func A. Under special circumstances State 1 transitions to State 2 when Func B requests the execution of Func D rather than Func C. State 2 may exists for only one cycle (D-E-exit) or many (D-E-D-E-…-exit), but

when exit occurs in this example, it does so without calling any function. By default then, the AppMgr's Idle function runs Func A since it is the entry point function.

FIG. 7 shows an example of a security system having features of the WSN described with respect to FIGS. 1 to 6 and having the various functionalities described herein. As shown in FIG. 7, correlation processing receives inputs from certain constrained nodes (although these can also be fully functional nodes). These inputs may include credential information and video information, and the correlation processing may produce correlated results that are sent over the network. Context management processing receives inputs from certain constrained nodes (although these can also be fully functional nodes) e.g., credential information and video and grouping information, and performs context processing with results sent over the network. The network supports operation of emergency exit indicators; emergency cameras as well as distributed rule processing and rule engine/messaging processing. Range extenders are used with e.g., gateways, and a real time location system receives inputs from various sensors (e.g., constrained type) as shown. Servers interface to the WSN via a cloud computing configuration and parts of some networks can be run as sub-nets.

The sensors provide in addition to an indication that something is detected in an area within the range of the sensors, detailed additional information that can be used to evaluate what that indication may be without the intrusion detection panel being required to perform extensive analysis of inputs to the particular sensor.

For example, a motion detector could be configured to analyze the heat signature of a warm body moving in a room to determine if the body is that of a human or a pet. Results of that analysis would be a message or data that conveys information about the body detected. Various sensors thus are used to sense sound, motion, vibration, pressure, heat, images, and so forth, in an appropriate combination to detect a true or verified alarm condition at the intrusion detection panel.

Recognition software can be used to discriminate between objects that are a human and objects that are an animal; further facial recognition software can be built into video cameras and used to verify that the perimeter intrusion was the result of a recognized, authorized individual. Such video cameras would comprise a processor and memory and the recognition software to process inputs (captured images) by the

camera and produce the metadata to convey information regarding recognition or lack of recognition of an individual captured by the video camera. The processing could also alternatively or in addition include information regarding characteristic of the individual in the area captured/monitored by the video camera. Thus, depending on the circumstances, the information would be either metadata received from enhanced motion detectors and video cameras that performed enhanced analysis on inputs to the sensor that gives characteristics of the perimeter intrusion or a metadata resulting from very complex processing that seeks to establish recognition of the object.

Sensor devices can integrate multiple sensors to generate more complex outputs so that the intrusion detection panel can utilize its processing capabilities to execute algorithms that analyze the environment by building virtual images or signatures of the environment to make an intelligent decision about the validity of a breach.

Memory stores program instructions and data used by the processor of the intrusion detection panel. The memory may be a suitable combination of random access memory and read-only memory, and may host suitable program instructions (e.g. firmware or operating software), and configuration and operating data and may be organized as a file system or otherwise. The stored program instruction may include one or more authentication processes for authenticating one or more users. The program instructions stored in the memory of the panel may further store software components allowing network communications and establishment of connections to the data network. The software components may, for example, include an internet protocol (IP) stack, as well as driver components for the various interfaces, including the interfaces and the keypad . Other software components suitable for establishing a connection and communicating across network will be apparent to those of ordinary skill.

Program instructions stored in the memory, along with configuration data may control overall operation of the panel.

The monitoring server includes one or more processing devices (e.g., microprocessors), a network interface and a memory (all not illustrated). The monitoring server may physically take the form of a rack mounted card and may be in

communication with one or more operator terminals (not shown). An example monitoring server is a SURGARD™ SG-System III Virtual, or similar system.

The processor of each monitoring server acts as a controller for each monitoring server , and is in communication with, and controls overall operation, of each server. The processor may include, or be in communication with, the memory that stores processor executable instructions controlling the overall operation of the monitoring server. Suitable software enable each monitoring server to receive alarms and cause appropriate actions to occur. Software may include a suitable Internet protocol (IP) stack and applications/clients.

Each monitoring server of the central monitoring station may be associated with an IP address and port(s) by which it communicates with the control panels and/or the user devices to handle alarm events, etc. The monitoring server address may be static, and thus always identify a particular one of monitoring server to the intrusion detection panels. Alternatively, dynamic addresses could be used, and associated with static domain names, resolved through a domain name service.

The network interface card interfaces with the network to receive incoming signals, and may for example take the form of an Ethernet network interface card (NIC). The servers may be computers, thin-clients, or the like, to which received data representative of an alarm event is passed for handling by human operators. The monitoring station may further include, or have access to, a subscriber database that includes a database under control of a database engine. The database may contain entries corresponding to the various subscriber devices/processes to panels like the panel that are serviced by the monitoring station.

All or part of the processes described herein and their various modifications (hereinafter referred to as "the processes") can be implemented, at least in part, via a computer program product, i.e., a computer program tangibly embodied in one or more tangible, physical hardware storage devices that are computer and/or machine-readable storage devices for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a

stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a network.

Actions associated with implementing the processes can be performed by one or more programmable processors executing one or more computer programs to perform the functions of the calibration process. All or part of the processes can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) and/or an ASIC (application-specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only storage area or a random access storage area or both. Elements of a computer (including a server) include one or more processors for executing instructions and one or more storage area devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from, or transfer data to, or both, one or more machine-readable storage media, such as mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks.

Tangible, physical hardware storage devices that are suitable for embodying computer program instructions and data include all forms of non-volatile storage, including by way of example, semiconductor storage area devices, e.g., EPROM, EEPROM, and flash storage area devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks and volatile computer memory, e.g., RAM such as static and dynamic RAM, as well as erasable memory, e.g., flash memory.

In addition, the logic flows depicted in the figures do not require the particular order shown, or sequential order, to achieve desirable results. In addition, other actions may be provided, or actions may be eliminated, from the described flows, and other components may be added to, or removed from, the described systems. Likewise, actions depicted in the figures may be performed by different entities or consolidated.

Elements of different embodiments described herein may be combined to form other embodiments not specifically set forth above. Elements may be left out of the processes, computer programs, Web pages, etc. described herein without adversely affecting their operation. Furthermore, various separate elements may be combined into one or more individual elements to perform the functions described herein.

Other implementations not specifically described herein are also within the scope of the following claims.

WHAT IS CLAIMED IS:

1. A networked sensor system comprises:

an upper tier of server devices, the server devices comprising:

processor devices; and

memory in communication with the processor devices;

a middle tier of gateway devices that are in communication with one or more of the upper tier server devices;

a lower level tier of devices that comprise fully functional sensor nodes with at least some of the fully functional sensor nodes including an application layer that executes routines to provide node sensor functions and an application layer manager to manage the application layer in the at least some of the functional nodes in the lower tier of devices.

2. The networked system of claim 1, wherein the lower tier fully-functional nodes are wireless devices and constrained wireless nodes or end-nodes that are sensors for a for physical intrusion detection/alarm monitoring system.

3. The networked system of claim 1, wherein the gateways are equipped with an access point where a function node is physically attached to that provides a wireless connection point to other nodes in the wireless network.

4. The networked system of claim 1, wherein the application layer manager includes a state machine comprised of sets of functions that execute in coordination, where functions can be individually deleted, substituted, or added unto in order to alter states in the state machine.

5. The networked system of claim 1, wherein the application layer manager uses an edge device operating system that allows for loading and execution of individual functions after the booting of the device or without rebooting the device.

6. The networked system of claim 1, wherein an application layer manager receives requests from state functions and in response handles pass-offs of functions currently operating to destination functions.

7. The networked system of claim 6, wherein the application layer manager accesses a transition table that stores a central descriptor for the state functions.

8. The networked system of claim 1, wherein the application layer manager is firmware and is isolated from other firmware modules in the lower tier devices.

9. A method of managing a networked sensor system, the method comprising:

partitioning the networked system into an upper tier of server devices, a middle tier of gateway devices that are in communication with upper tier servers and a lower level tier of sensor devices that are fully functional nodes with at least some of the functional nodes including an application layer that execute routines to provide sensor node functions; and

managing the lower tier of devices through device executed program managers that execute state machines to control application layers in each of the at least some of the functional nodes.

10. The method of claim 9, wherein the lower tier fully-functional nodes are wireless devices and constrained wireless nodes or end-nodes that are sensors for a for physical intrusion detection/alarm monitoring system.

11. The method of claim 9, wherein the state machine is comprised of sets of functions that execute in coordination, where functions can be individually deleted, substituted, or added unto in order to alter states in the state machine.

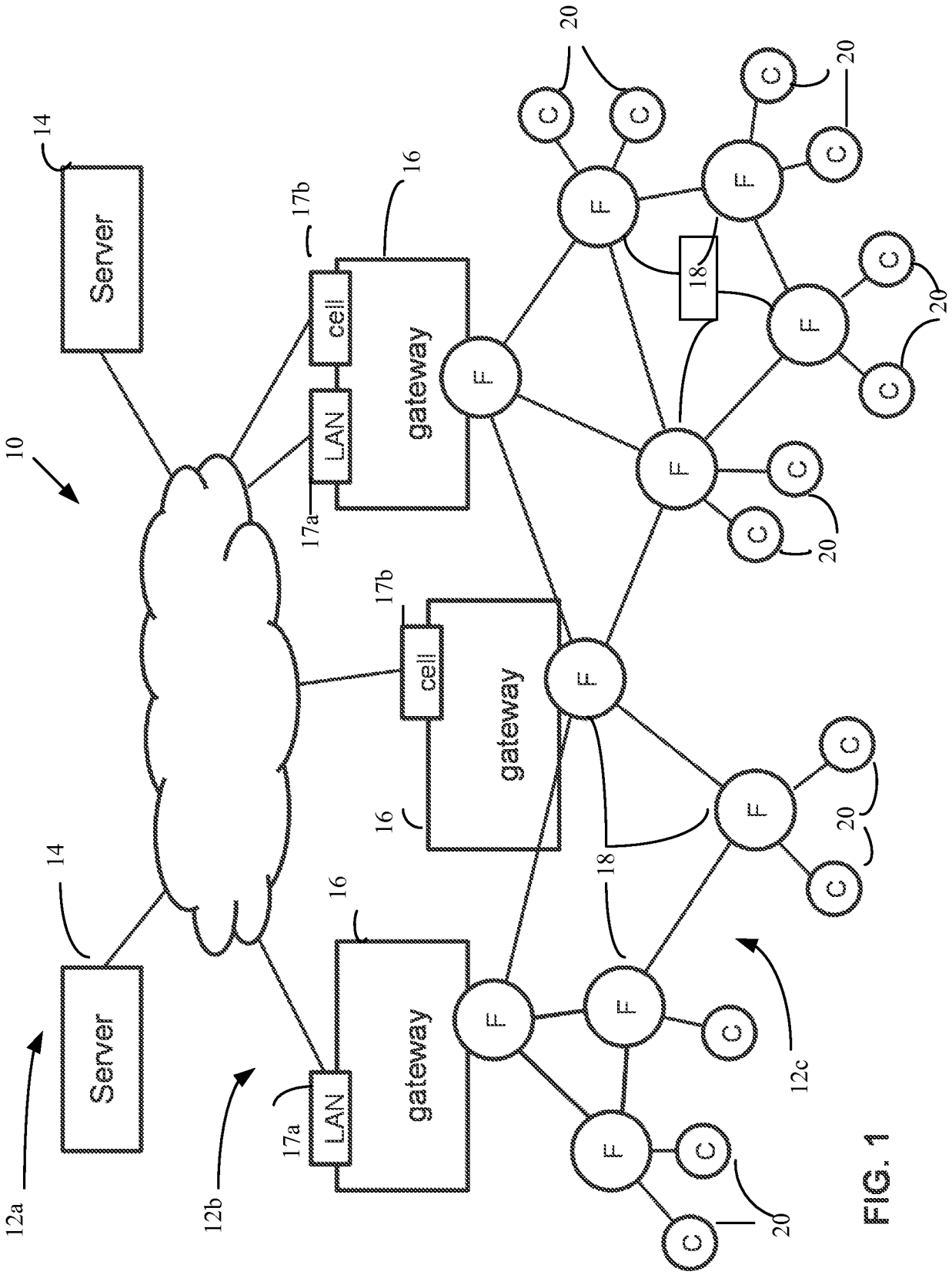

Server
Server
gateway
gateway
gateway
LAN
LAN
cell
cell
F
C
10
12a
12b
12c
14
16
17a
17b
18
20

FIG. 1

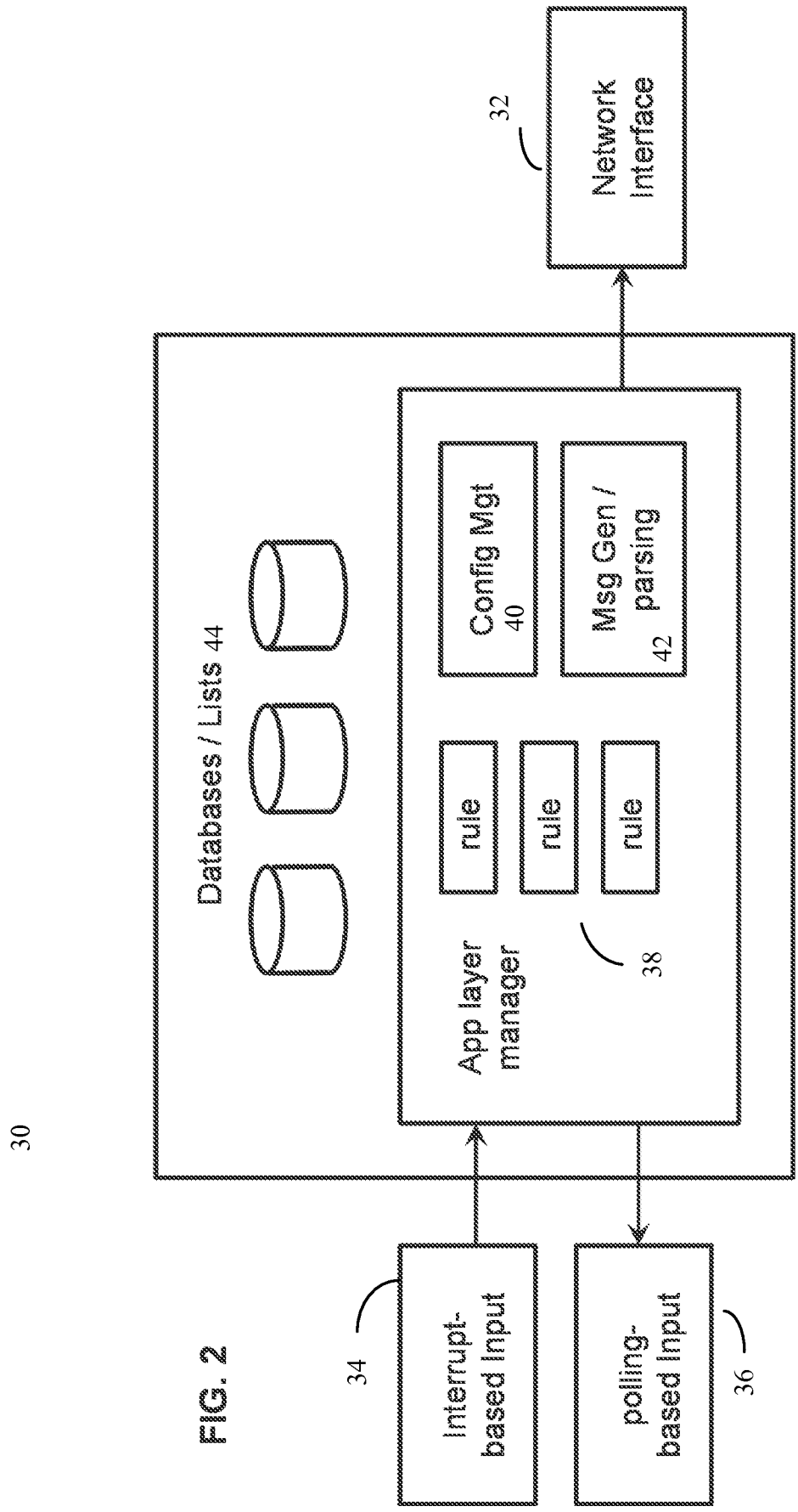

30
Databases / Lists 44
App layer manager
38
rule
rule
rule
Config Mgt 40
Msg Gen / parsing 42
Network Interface
32
Interrupt-based Input
34
polling-based Input
36

FIG. 2

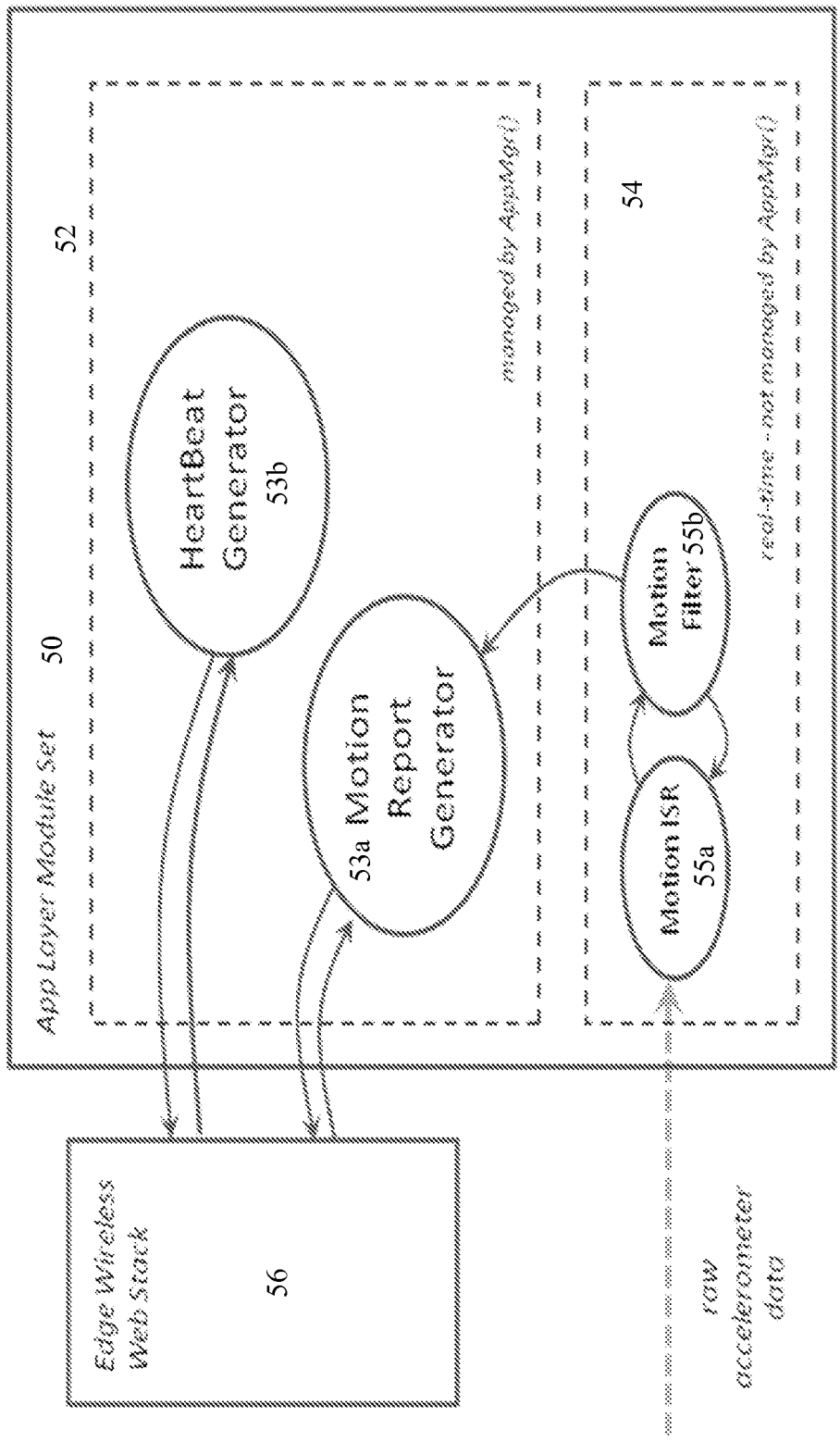

App Layer Module Set
50
52
HeartBeat Generator 53b
53a Motion Report Generator
managed by AppMgr()
54
Motion Filter 55b
Motion ISR 55a
real-time - not managed by AppMgr()
Edge Wireless Web Stack
56
raw accelerometer data

FIG. 3

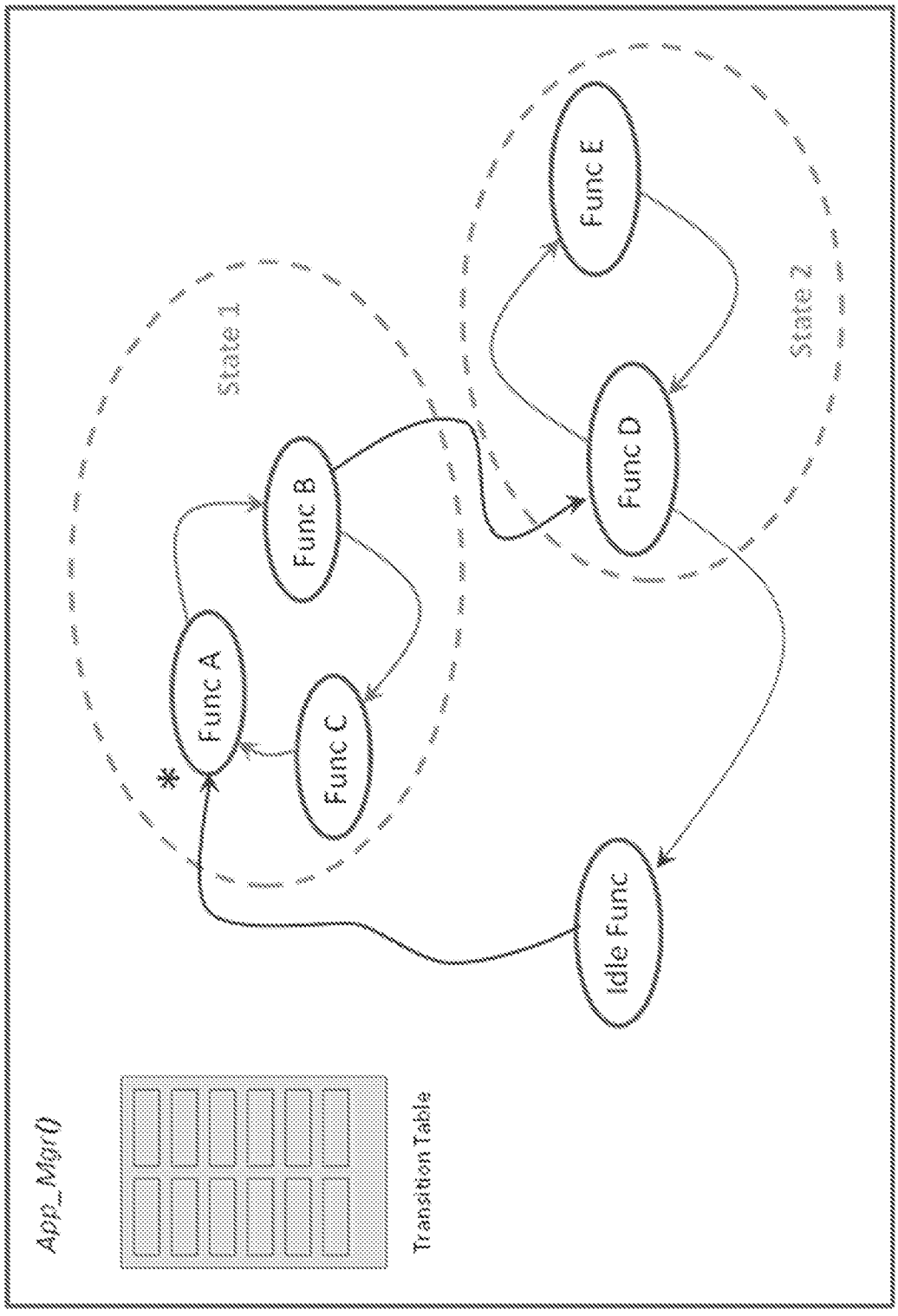

App_Mgr()
Transition Table
State 1
State 2
Func A
Func B
Func C
Func D
Func E
Idle Func
*

FIG. 4

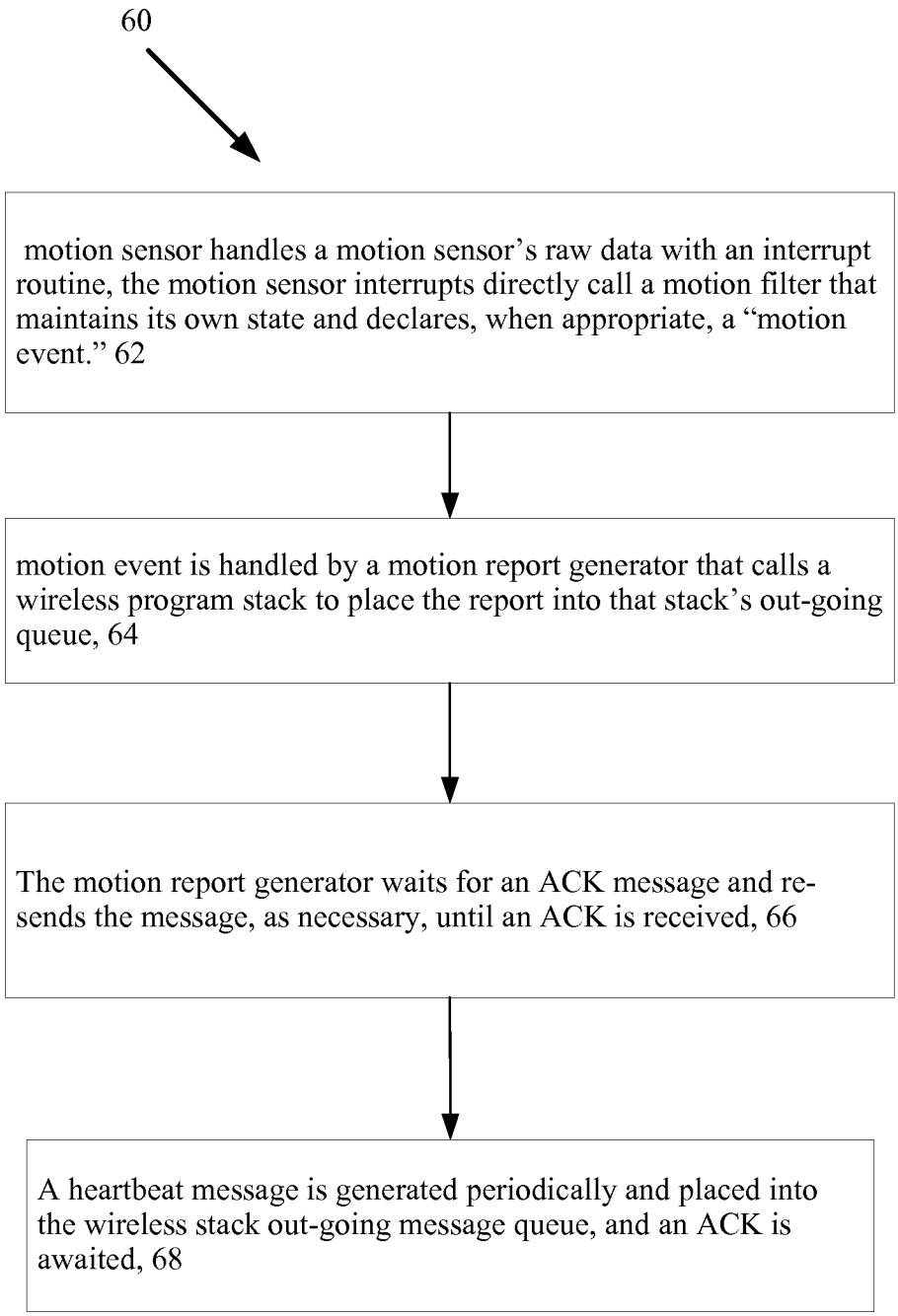

60
motion sensor handles a motion sensor's raw data with an interrupt routine, the motion sensor interrupts directly call a motion filter that maintains its own state and declares, when appropriate, a "motion event." 62
motion event is handled by a motion report generator that calls a wireless program stack to place the report into that stack's out-going queue, 64
The motion report generator waits for an ACK message and re-sends the message, as necessary, until an ACK is received, 66
A heartbeat message is generated periodically and placed into the wireless stack out-going message queue, and an ACK is awaited, 68

FIG. 5

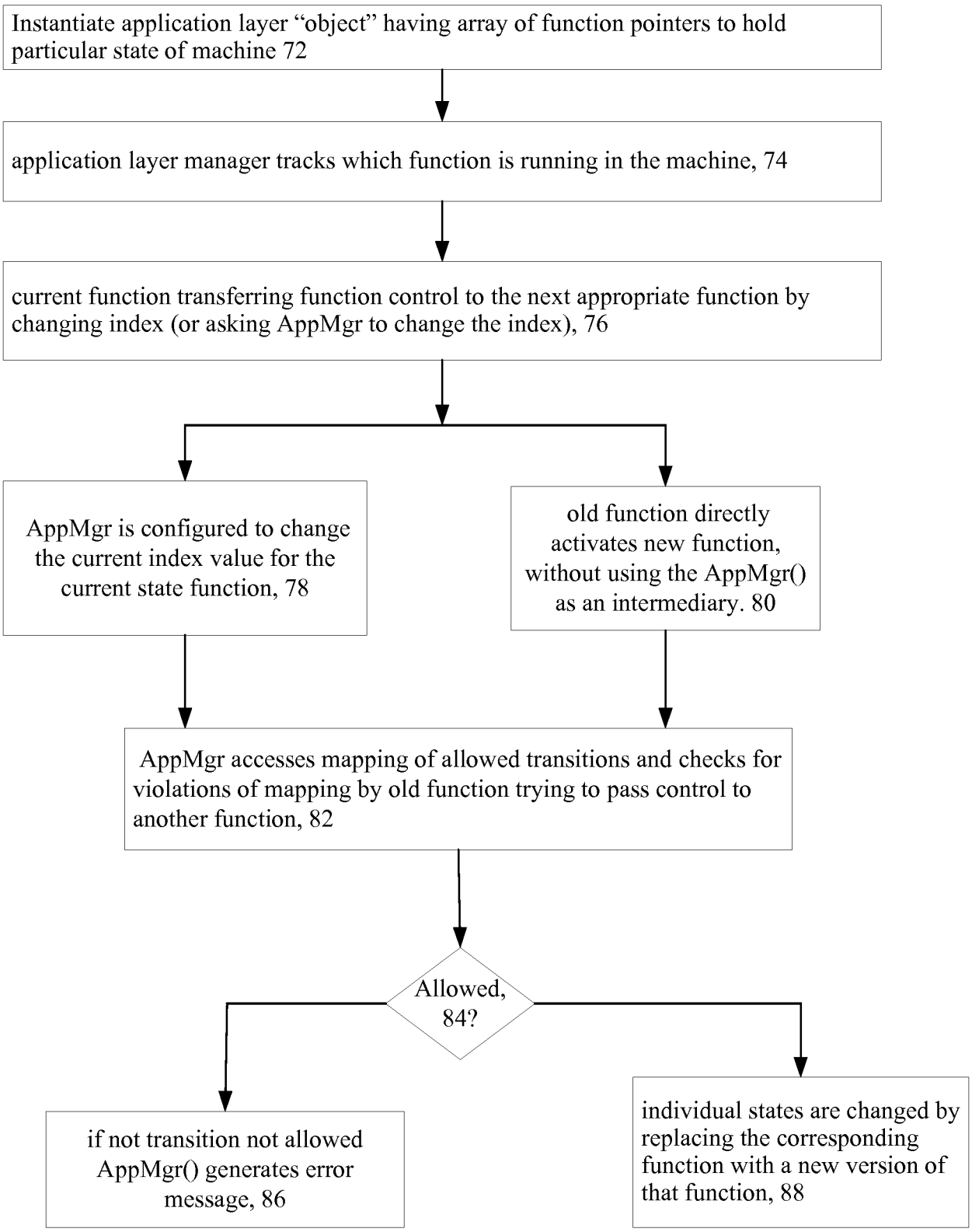

Instantiate application layer "object" having array of function pointers to hold particular state of machine 72
application layer manager tracks which function is running in the machine, 74
current function transferring function control to the next appropriate function by changing index (or asking AppMgr to change the index), 76
AppMgr is configured to change the current index value for the current state function, 78
old function directly activates new function, without using the AppMgr() as an intermediary. 80
AppMgr accesses mapping of allowed transitions and checks for violations of mapping by old function trying to pass control to another function, 82
Allowed, 84?
if not transition not allowed AppMgr() generates error message, 86
individual states are changed by replacing the corresponding function with a new version of that function, 88

FIG. 6

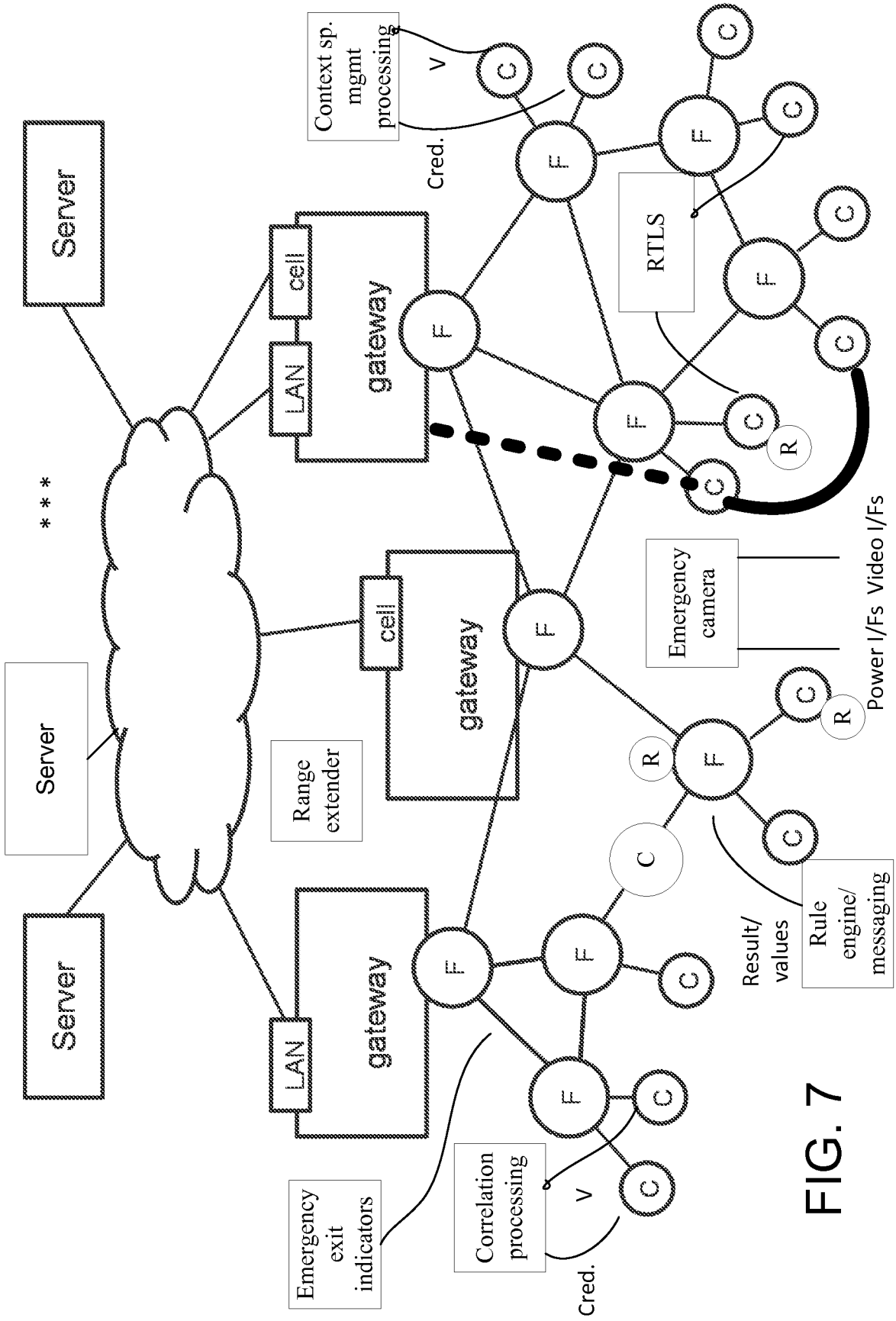

Server
Server
Server
* * *
LAN
cell
gateway
LAN
cell
gateway
gateway
Range extender
Context sp. mgmt processing
V
Cred.
RTLS
Emergency camera
Power I/Fs
Video I/Fs
Result/ values
Rule engine/ messaging
Emergency exit indicators
Correlation processing
V
Cred.
F
C
R

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/017212

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G08B 25/10 (2015.01)
CPC - G08B 25/10 (2015.01)
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8) - G08B 25/10 (2015.01)
USPC - 340/521, 539.22, 541; 709/223, 224

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
CPC - G08B 25/009, 016, 10 (2015.01) (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Orbit, Google Patents, Google Scholar.
Search terms used: wireless, sensor, network, intrusion, detection, alarm, application layer, nodes, gateway, server, tier, firmware

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	US 2008/0291017 A1 (YERMAL et al) 27 November 2008 (27.11.2008) entire document	1, 3-4, 6, 8-9, 11 --- 2, 5, 7, 10
Y	US 2004/0090329 A1 (HITT) 13 May 2004 (13.05.2004) entire document	2, 10
Y	US 2013/0239192 A1 (LINGA et al) 12 September 2013 (12.09.2013) entire document	5
Y	US 5,414,812 A (FILIP et al) 09 May 1995 (09.05.1995) entire document	7
A	US 2011/0102171 A1 (RAJI et al) 05 May 2011 (05.05.2011) entire document	1-11
A	US 8,487,762 B1 (MCMULLEN et al) 16 July 2013 (16.07.2013) entire document	1-11
A	US 2008/0068150 A1 (NGUYEN et al) 20 March 2008 (20.03.2008) entire document	1-11

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
24 April 2015	02 JUN 2015

Name and mailing address of the ISA/US	Authorized officer:
Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)