US 20030135547A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2003/0135547 A1

Kent et al. (43) Pub. Date: Jul. 17, 2003

(54) EXTENSIBLE MODULAR COMMUNICATION EXECUTIVE WITH ACTIVE MESSAGE QUEUE AND INTELLIGENT MESSAGE PRE-VALIDATION

(76) Inventors: **J. Thomas Kent**, Cincinnati, OH (US); **David A. Toot**, Mason, OH (US)

Correspondence Address:
Steven J. Trzaska
Congins Corporation Law Department
2500 Renaissance Boulevard
Suite 200
Gulph Mills, PA 19406 (US)

(21) Appl. No.: **10/188,853**

(22) Filed: **Jul. 5, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/307,347, filed on Jul. 23, 2001. Provisional application No. 60/307,348, filed on Jul. 23, 2001.

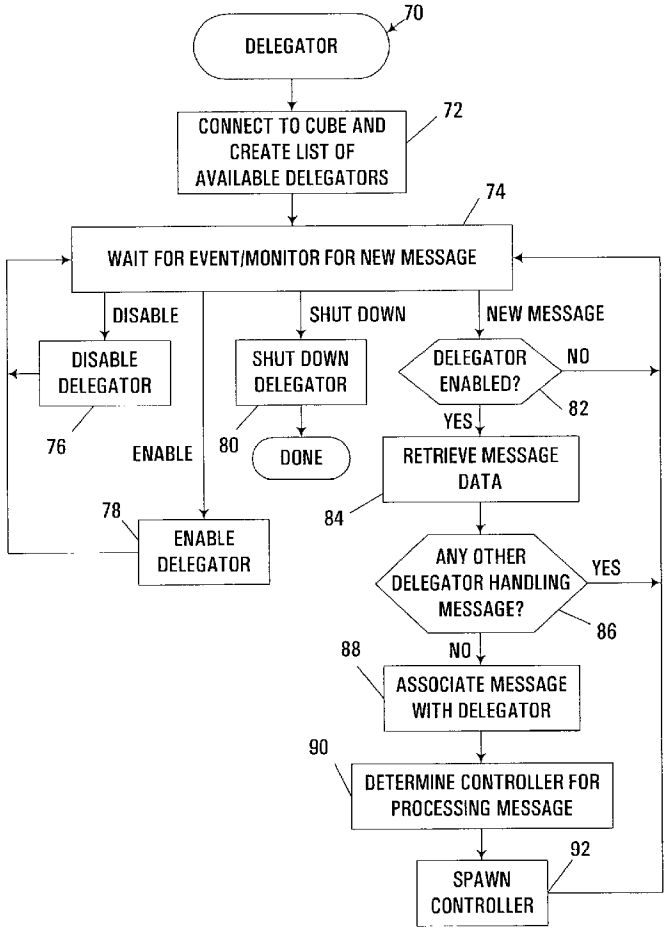
Publication Classification

(51) **Int. Cl.**[7] **G06F 15/16**

(52) **U.S. Cl.** **709/203**

(57) **ABSTRACT**

An apparatus, program product and method incorporate an extensible modular communication executive for use integrating one or more electronic devices with one another with reduced customization overhead. A modular architecture is used to facilitate message-based communications in such a manner that queuing strategies, business rules and the like may be accommodated within a message-based environment in a reliable and efficient manner. Through the use of a modular architecture, application-specific software components can be assembled together to readily adapt a generic message-based system for use in a specific application. Moreover, intelligent pre-validation of messages may be implemented in such a modular architecture to permit a business rule-independent messaging infrastructure to be readily adapted to support specific business rule requirements for a particular application.



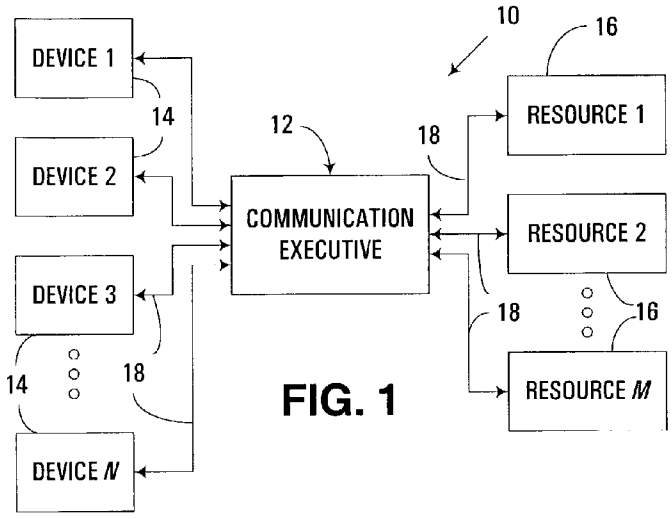


FIG. 1

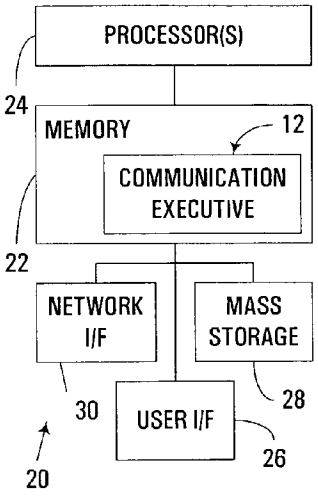


FIG. 2

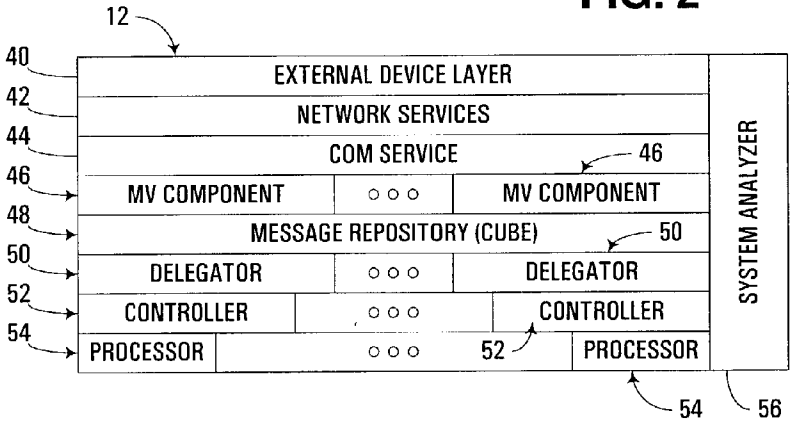


FIG. 3

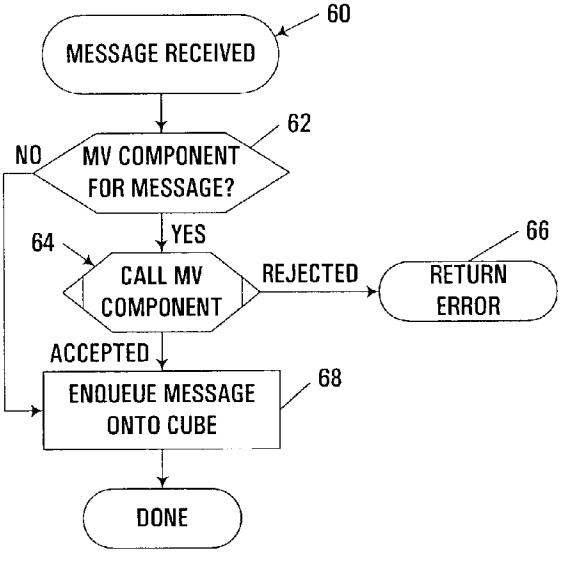


FIG. 4

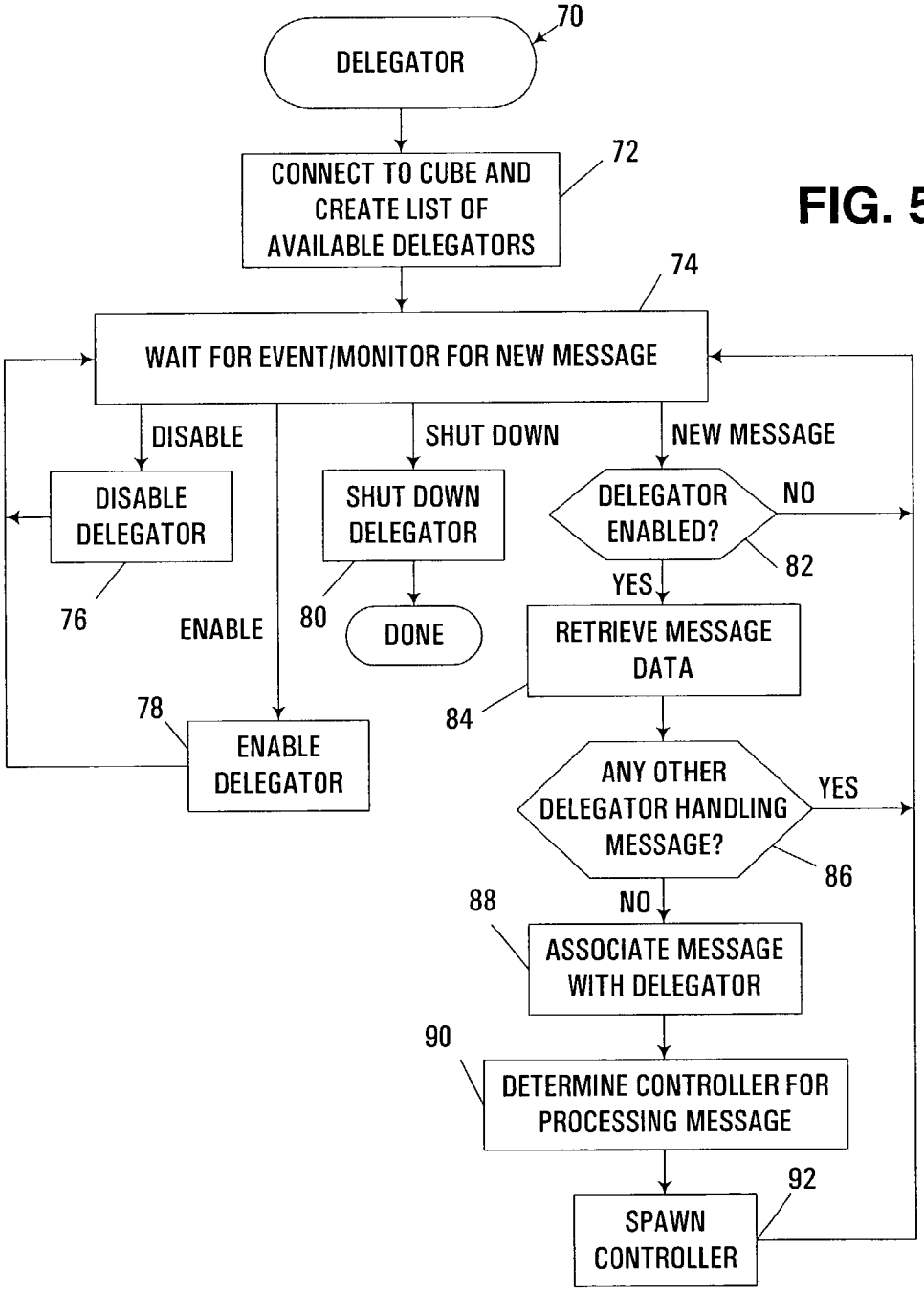

DELEGATOR
70
CONNECT TO CUBE AND CREATE LIST OF AVAILABLE DELEGATORS
72
74
WAIT FOR EVENT/MONITOR FOR NEW MESSAGE
DISABLE
SHUT DOWN
NEW MESSAGE
DISABLE DELEGATOR
76
ENABLE
SHUT DOWN DELEGATOR
80
DONE
DELEGATOR ENABLED?
NO
82
YES
RETRIEVE MESSAGE DATA
84
78
ENABLE DELEGATOR
ANY OTHER DELEGATOR HANDLING MESSAGE?
YES
86
88
NO
ASSOCIATE MESSAGE WITH DELEGATOR
90
DETERMINE CONTROLLER FOR PROCESSING MESSAGE
92
SPAWN CONTROLLER

FIG. 5

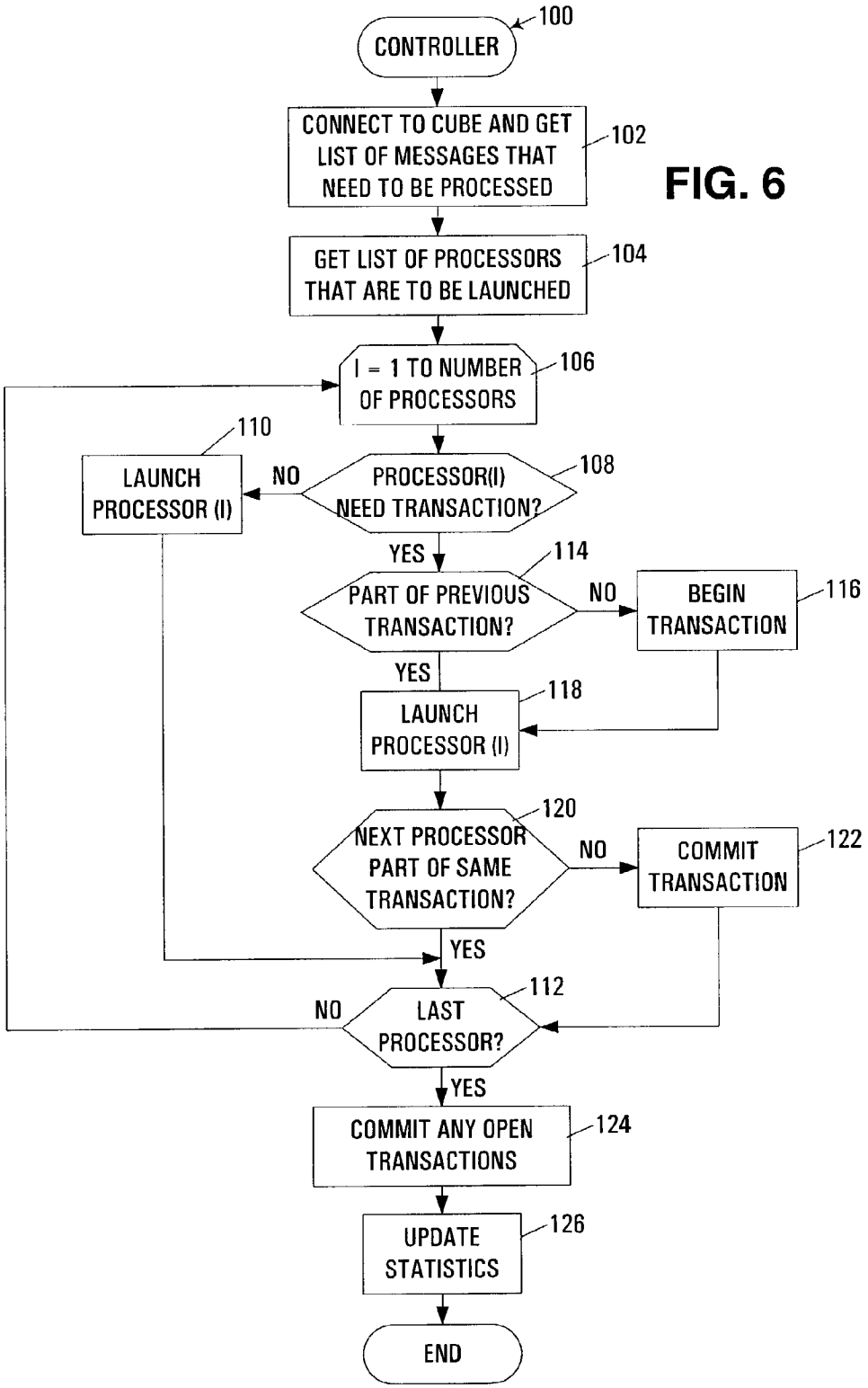

100
CONTROLLER
CONNECT TO CUBE AND GET LIST OF MESSAGES THAT NEED TO BE PROCESSED
102
GET LIST OF PROCESSORS THAT ARE TO BE LAUNCHED
104
I = 1 TO NUMBER OF PROCESSORS
106
PROCESSOR(I) NEED TRANSACTION?
108
NO
LAUNCH PROCESSOR (I)
110
YES
PART OF PREVIOUS TRANSACTION?
114
NO
BEGIN TRANSACTION
116
YES
LAUNCH PROCESSOR (I)
118
NEXT PROCESSOR PART OF SAME TRANSACTION?
120
NO
COMMIT TRANSACTION
122
YES
NO
LAST PROCESSOR?
112
YES
COMMIT ANY OPEN TRANSACTIONS
124
UPDATE STATISTICS
126
END

FIG. 6

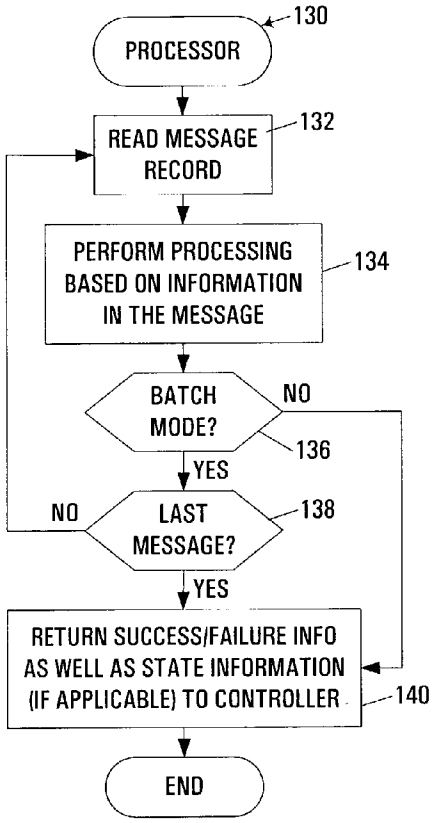


FIG. 7

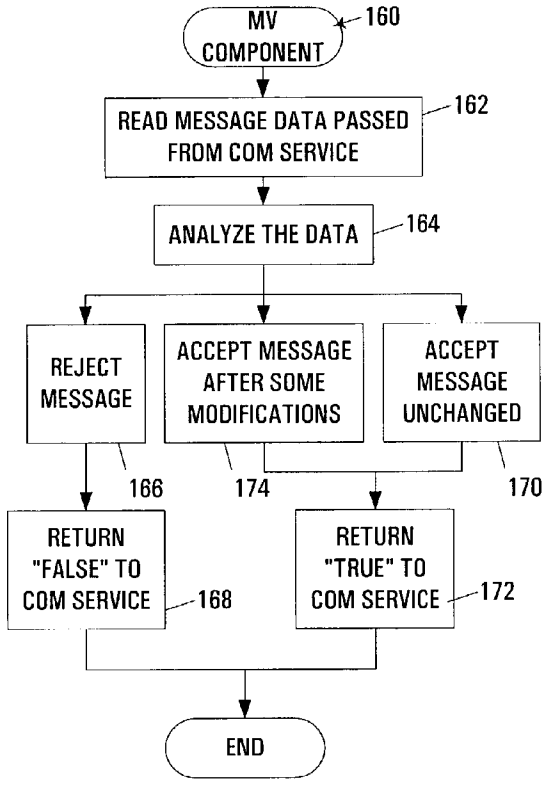


FIG. 9

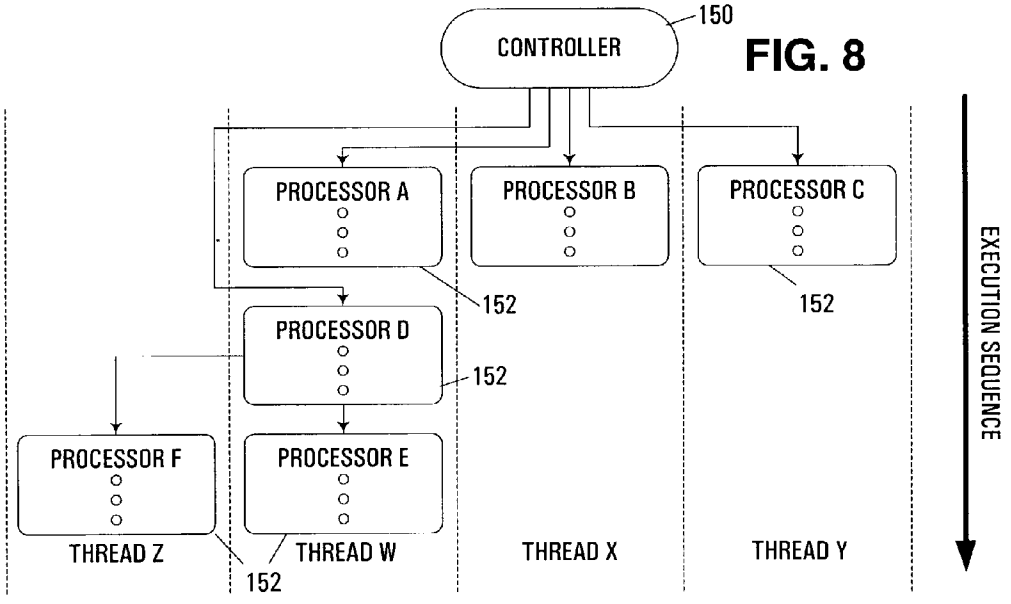


FIG. 8

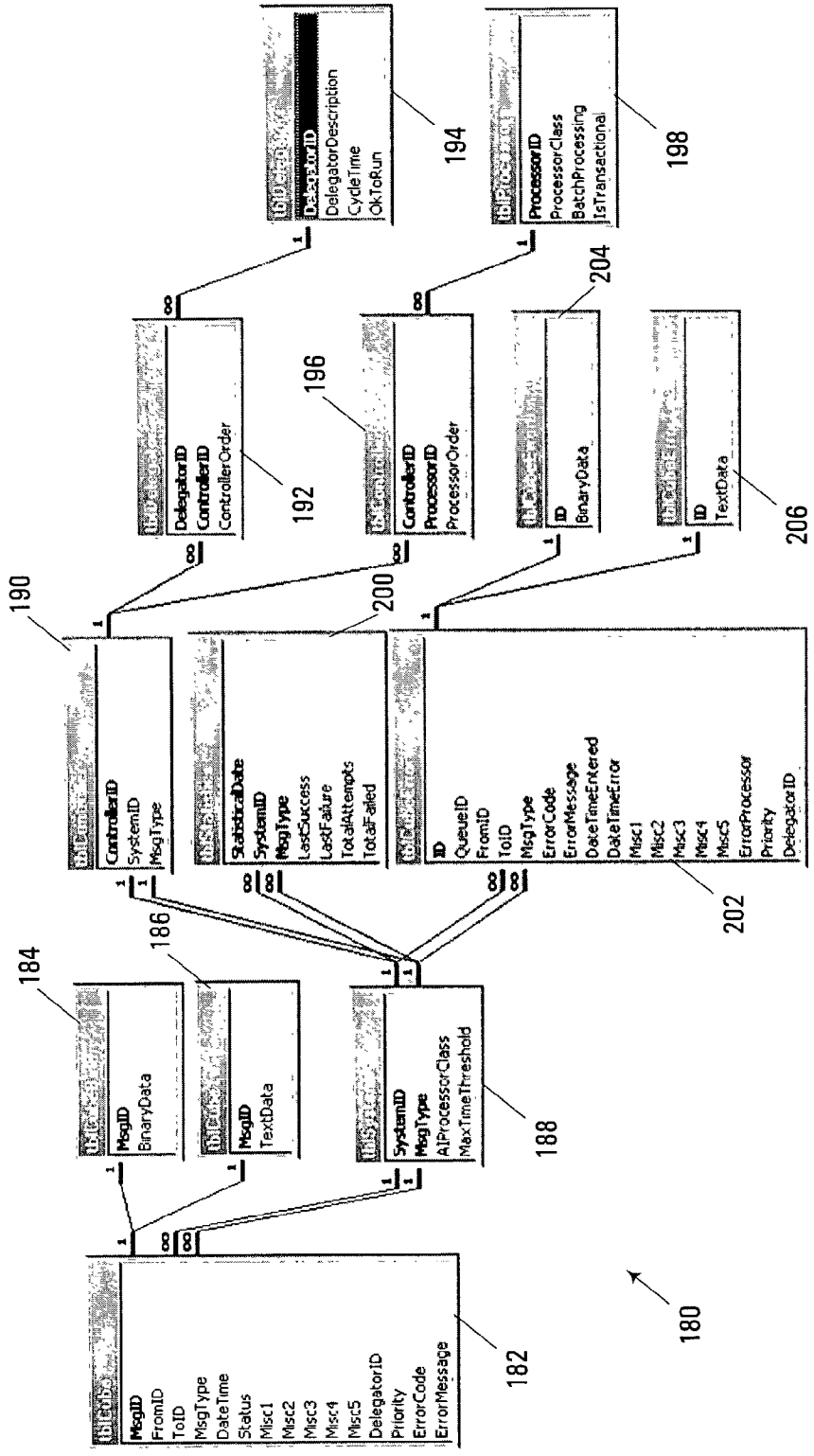

MsgID
FromID
ToID
MsgType
DateTime
Status
Misc1
Misc2
Misc3
Misc4
Misc5
DelegatorID
Priority
ErrorCode
ErrorMessage
182
180
MsgID
BinaryData
184
MsgID
TextData
186
SystemID
MsgType
AlProcessorClass
MaxTimeThreshold
188
ControllerID
SystemID
MsgType
190
StatisticalDate
SystemID
MsgType
LastSuccess
LastFailure
TotalAttempts
TotalFailed
200
ID
QueueID
FromID
ToID
MsgType
ErrorCode
ErrorMessage
DateTimeEntered
DateTimeError
Misc1
Misc2
Misc3
Misc4
Misc5
ErrorProcessor
Priority
DelegatorID
202
DelegatorID
ControllerID
ControllerOrder
192
ControllerID
ProcessorID
ProcessorOrder
196
ID
BinaryData
204
ID
TextData
206
DelegatorID
DelegatorDescription
CycleTime
OkToRun
194
ProcessorID
ProcessorClass
BatchProcessing
IsTransactional
198

FIG. 10

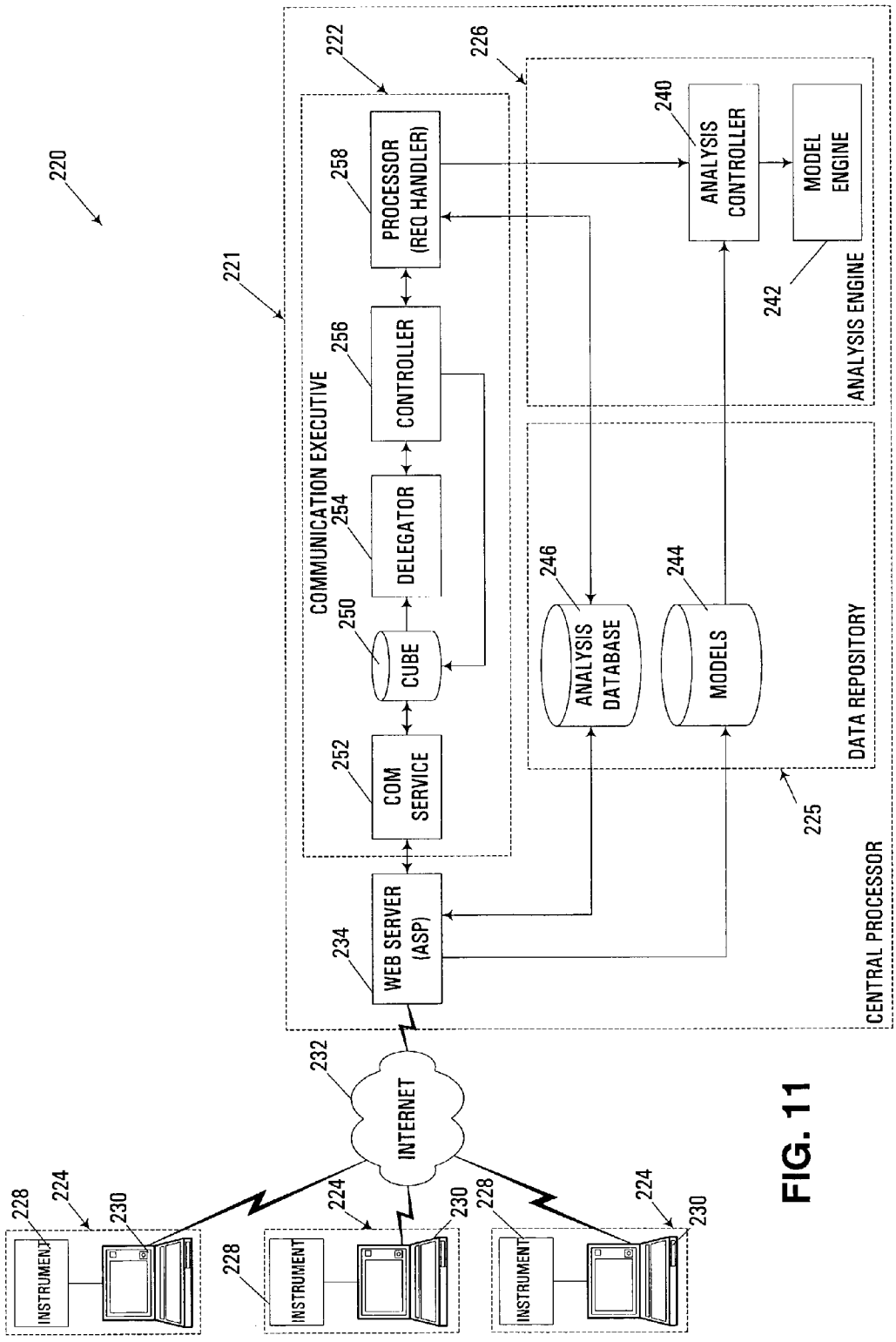

220
221
222
226
240
ANALYSIS CONTROLLER
MODEL ENGINE
242
ANALYSIS ENGINE
258
PROCESSOR (REQ HANDLER)
256
CONTROLLER
254
DELEGATOR
250
CUBE
252
COM SERVICE
COMMUNICATION EXECUTIVE
246
ANALYSIS DATABASE
244
MODELS
DATA REPOSITORY
225
234
WEB SERVER (ASP)
CENTRAL PROCESSOR
232
INTERNET
228
224
230
INSTRUMENT

FIG. 11

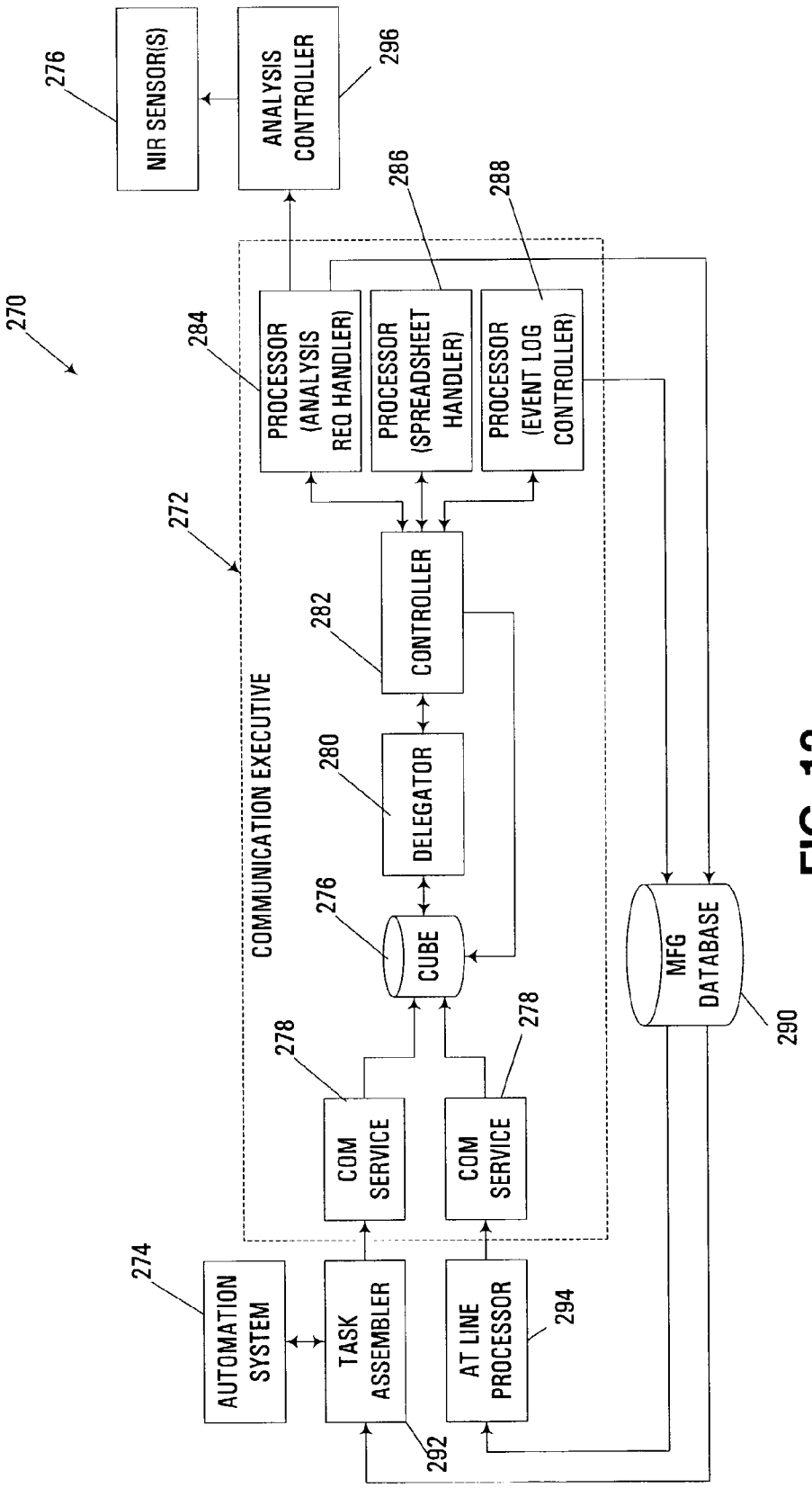

270
272
274
276
278
280
282
284
286
288
290
292
294
296
NIR SENSOR(S)
ANALYSIS CONTROLLER
PROCESSOR (ANALYSIS REQ HANDLER)
PROCESSOR (SPREADSHEET HANDLER)
PROCESSOR (EVENT LOG CONTROLLER)
COMMUNICATION EXECUTIVE
CONTROLLER
DELEGATOR
CUBE
COM SERVICE
COM SERVICE
MFG DATABASE
AUTOMATION SYSTEM
TASK ASSEMBLER
AT LINE PROCESSOR

FIG. 12

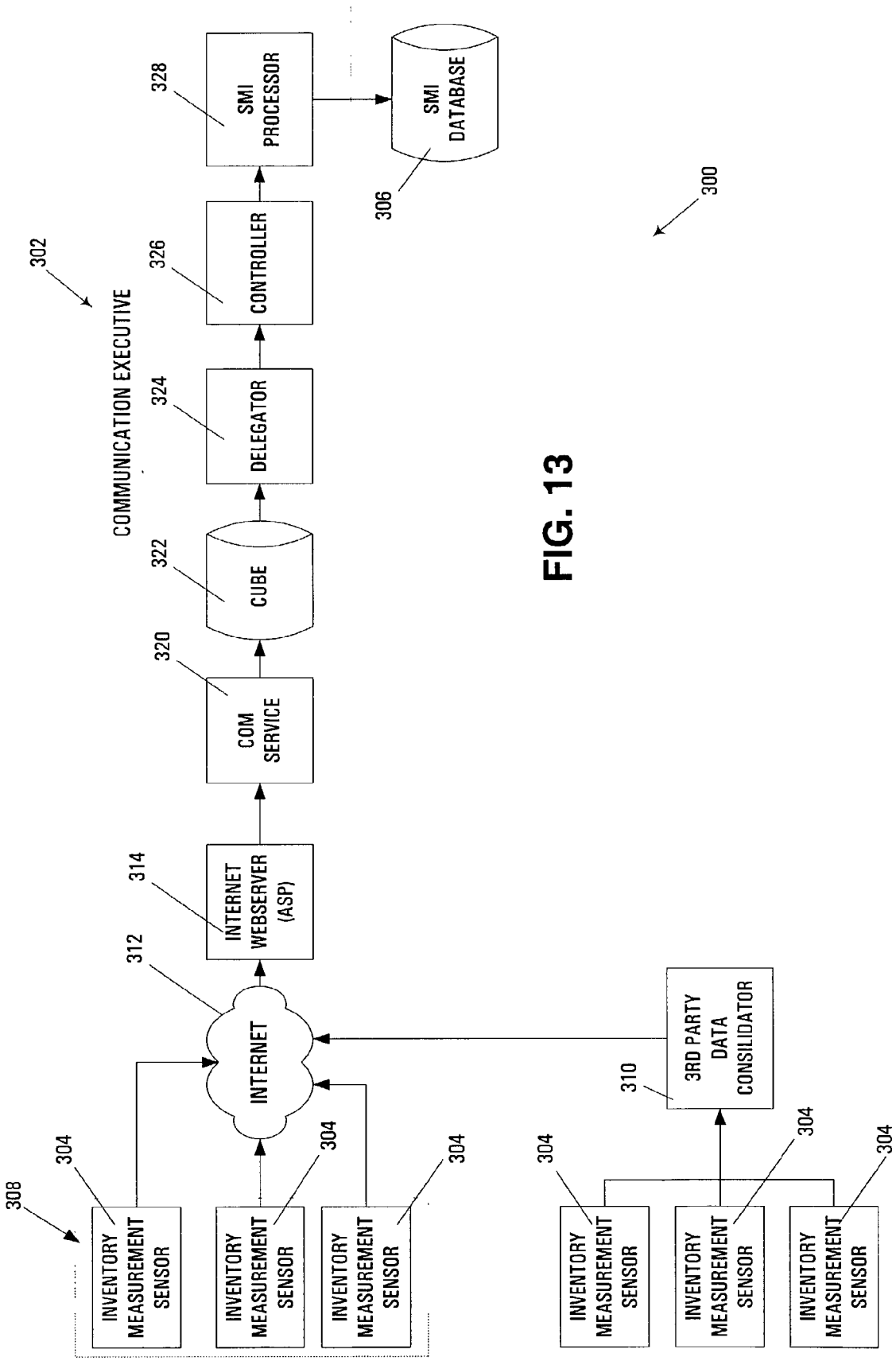

300
302
COMMUNICATION EXECUTIVE
328
SMI PROCESSOR
SMI DATABASE
306
326
CONTROLLER
324
DELEGATOR
322
CUBE
320
COM SERVICE
314
INTERNET WEBSERVER (ASP)
312
INTERNET
3RD PARTY DATA CONSILIDATOR
310
308
304
INVENTORY MEASUREMENT SENSOR

FIG. 13

EXTENSIBLE MODULAR COMMUNICATION EXECUTIVE WITH ACTIVE MESSAGE QUEUE AND INTELLIGENT MESSAGE PRE-VALIDATION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of two U.S. Provisional Patent Applications, Serial Nos. 60/307,347 and 60/307,348, both filed on Jul. 23, 2001, the disclosures of which are incorporated by reference herein. This application is also related to U.S. patent application Ser. No. _____ , filed on even date herewith by Alan P. Lundstedt, et al., and entitled "ON-SITE ANALYSIS SYSTEM WITH CENTRAL PROCESSOR AND METHOD OF ANALYZING," the disclosure of which is also incorporated by reference herein.

FIELD OF THE INVENTION

[0002] The present invention relates to managing communications between sending and receiving electronic devices, and in particular, to message-based communications for interfacing independent electronic devices.

BACKGROUND OF THE INVENTION

[0003] Data communication is an important aspect of any data processing system. Within integrated circuit chips such as microprocessors, communication of data between various functional units is required when performing processing tasks. Likewise, microprocessors require communication with external devices, e.g., to receive input from a user or display input from a user, as well as to store or retrieve from a memory or storage device. Computers and other electronic devices also have a need to communicate with other such devices, e.g., to distribute processing tasks among multiple devices or to exchange information managed by one device with another device that needs to utilize the information.

[0004] As an example, in a manufacturing control system, various control devices such as machines, valves, actuators, etc. are typically managed by a controller, which receives feedback from one or more sensing devices that monitor the progress of a manufacturing operation. In a chemical processing environment, for example, there are control devices such as control valves that add ingredients to a chemical formulation or that transfer ingredients or finished products to and from storage tanks, environmental controls that control reaction temperatures, ambient pressures, and the like, etc. Morever, in such an environment there are sensing devices such as temperature gauges, pressure gauges, conductivity sensors, etc. When managed by a central controller, information must be communicated from sensing devices to the controller, and control signals must be output to the various control devices to manage the process according to desired process parameters. Thus, data communication is required to communicate input data, output control signals, output status information, etc. Moreover, in many situations, such data communication must be as near to "real time" as possible, particularly where safety or product quality can be affected by responsiveness.

[0005] Whereas early manufacturing control systems relied on simple analog and/or digital control wires between relatively primitive sensors, relay switches, display indicators, etc., as technology has advanced, the sensors, displays, control devices and the like have become more sophisticated, whereby the manner of communication between such devices has become more complex and comprehensive. In particular, many sensing and control devices have incorporated processing logic that often reduces the amount of interaction required with a central controller, and thus permits a central controller to handle higher level tasks in a substantially more efficient manner. One or more networks, each representing point-to-point or shared electronic interconnects, are typically used to interface the devices to the controller.

[0006] Some manufacturing control systems are highly centralized, whereby sensing devices gather data and forward the data to a central controller that processes the data and controls one or more control devices accordingly. However, it has been found that a more decentralized approach is often superior in many situations, particularly where sensing and control devices may be physically dispersed over a wide geographical area. To implement a decentralized system, often one or more controllers are interfaced together, with individual controllers utilized to interface with selected subsets of sensing and/or control devices in the system.

[0007] In decentralized systems, communications between controllers, sensing devices and control devices are typically more sophisticated than centralized systems that utilize central controllers and relatively "dumb" sensing and/or control devices. For example, message-based communications are often used in decentralized systems, whereby information units known as messages are passed between devices using a particular format that is understood by both the sending and receiving devices. Messages are often processed as "events" that are capable of being responded to by a receiving device with a similar type of message. Messages may also be ordered to ensure timely delivery and eliminate race conditions that could compromise data integrity or result in incorrect control settings.

[0008] Another aspect of a decentralized system is the ability to incorporate a wide variety of devices into a single system, often using devices that are manufactured by different entities and that may not be specifically adapted to work with other types of devices also used in the system. In many instances, such devices are specifically configured to communicate information according to a particular communications protocol, which may include a specific message format, and even a particular type of electrical connector and/or wiring. Should different devices within a particular system not subscribe to the same communications protocol, often custom hardware and/or software is required to appropriately interface the devices.

[0009] In addition, different devices may have different performance capabilities, e.g., response time, processing speed, memory capacity, etc., and thus the responsiveness of certain devices may not be adequate for other devices that require relatively short response times. Moreover, given the nature of many manufacturing processes, and the criticality of the controls to the success of the processes, communication must often be as close as possible to real time to maintain safety and quality control.

[0010] A number of different architectures are utilized in the art to implement manufacturing control systems of the types described above. For example, one type of manufacturing control system utilizes a central database that receives

input data from one or more sensing devices and stores the states of the input data in the database. Software scripts run on the database, analyzing the current input data and generating control signals that are output to one or more manufacturing control devices based upon set points established for various input data combinations.

[0011] Given the wide variety of sensors and control devices that may be required in a given manufacturing control system, typically a substantial amount of custom hardware and/or software is required to enable a database to both receive sensing device data and output appropriate control information to control devices. Moreover, despite the additional functionality that is often allocated to the sensing and control devices, such a system is still principally centralized since the status of the sensing device data and the control signals are maintained within the database. In such a system, typically the sensing devices and control devices are not capable of communicating directly with one another.

[0012] Other manufacturing control systems may be somewhat more decentralized, whereby relatively complex controllers are interfaced with a centralized database, with each controller responsible for a relatively greater number of sensing and/or control devices. Nonetheless, a central database is still utilized to maintain input data and output the control signals responsive thereto, and little or no direct communication between controllers, sensing devices and control devices is permitted.

[0013] Furthermore, as performance and complexity of the devices used in manufacturing control systems improve, maintaining "real time" operability becomes more and more difficult. A necessity for real-time response is especially problematic when complex devices from different manufacturers are utilized in the same system. Often a significant amount of custom hardware and/or software is required for each particular application. As a result, implementation time, effort, and expense are often significant. Preexisting systems are rarely, if ever, adaptable for use in other environments without substantial modification.

[0014] In other systems, it may also be desirable to utilize more sophisticated types of sensing and/or control devices that have conventionally not been integrated into manufacturing control environments. For example, infrared (IR) and near infrared (NIR) spectroscopy has been used in laboratory environments to perform quantitative analysis of chemical compositions. Spectroscopy provides the advantage that material samples can be analyzed relatively quickly and often without any preparation of the samples or any time-consuming wet chemistry. Many IR and NIR spectrum analyzers are principally designed to be used in laboratory environments, and traditionally, these types of laboratory instruments have not been particularly well-suited for use in manufacturing control systems.

[0015] Much of the incompatibility associated with such devices stems from difficulties communicating data with other components in a manufacturing control system, due to the fact that most commercially-available IR and NIR spectrum analyzers are not specifically designed for such applications. As a result, traditionally a manufacturing environment that relies on an IR or NIR spectrum analyzer requires that product samples be manually taken and physically brought to an analyzer for processing. In some environments, remote probes have been used within manufacturing equipment; however, substantial custom software and/or hardware is usually required to transmit the results generated by the probe to an analyzer for processing. Regardless of how the sample data is obtained, however, typically no control or direct communication is provided between an IR or NIR spectrum analyzer and the other components in a manufacturing control system.

[0016] IR and NIR spectrum analyzers also have other potential uses that would benefit from enhanced communication abilities. In particular, a spectrum analyzer typically must incorporate some form of analysis engine to process the spectral information generated by an NIR or IR sensor. Often, such an analyzer utilizes chemometrics or other multivariate regression methods to derive quantitative information about a particular sample from the sensor data. It has been found that a large number of outside factors, including various measurement conditions such as environmental parameters, instrument parameters and analysis techniques, as well as various sample conditions based upon the properties of the composition when the sensor data is obtained, can affect quantitative analysis results. For this reason, analysis is often a complex mathematical process requiring a relatively high performance computing system to process sensor data in a reliable and efficient manner. However, in a system whereby samples may require analysis in a large number of geographically dispersed locations, often analysis must be performed locally at each location, requiring costly and complex computer equipment at each location. Otherwise, if a central analysis system is used, communication incompatibilities complicate integration and adversely affect responsiveness.

[0017] In each of the foregoing environments, differences in the communication, performance and/or response requirements of different devices present challenges when such devices are being integrated into a common system. As one example, where message-based communications are used to interface devices, message buffers or "queues" are often used to temporarily store messages as they are being processed. Messages received by a device are typically enqueued on a message queue, and a computer program periodically polls the queue for new messages. Whenever a new message is detected, the computer program processes the message as appropriate. Messages are typically ordered for first-in-first-out processing, and in some instances prioritization rules may be applied to handle messages with differing degrees of urgency.

[0018] Queues are often associated with queuing "strategies," which essentially define how and by what means messages are received, stored, processed and ultimately responded to. These queuing strategies may ultimately affect response speed, processing speed, and other determining performance factors in both sending and receiving devices. As a result, when different devices are integrated into a message-based system, the queuing strategy used to interface those devices often must be specifically tailored to adequately handle the performance requirements of such devices. Where different devices have widely divergent performance characteristics, integration of the devices through a common queuing strategy can be difficult.

[0019] Moreover, many message-based systems inherently support various forms of "business rules" that define the accepted format of messages, as well as what messages

and message types are appropriate for a particular system. Typically, such rules are implemented by whatever computer program handles a received message, e.g., by performing error checking and the like while handling the message. The business rules are often highly application-specific. In addition, the business rules are often applied only after a message is enqueued on a queue, so that unacceptable messages will still typically be placed on a queue before any problems are detected.

[0020] Therefore, given the possible variations in different message-based communications systems, queuing strategies and business rules often must be specifically developed and implemented within individual devices in a particular system, often using custom hardware and/or software. As a result, integrating multiple disparate devices together into a system may not be possible in some instances, or in the least, may require substantial customization, leading to increased cost and reduced performance.

[0021] A need therefore exists in the art for a simplified manner of integrating disparate electronic devices together in a unified communications system, especially for a manner in which the degree of required customization is minimized. Moreover, a significant need exists for a manner of facilitating the integration of devices in particular into a message-based communications system so as to efficiently and reliably incorporate desirable queuing strategies and business rules with minimal customization.

SUMMARY OF THE INVENTION

[0022] The invention addresses these and other problems associated with the prior art by providing an apparatus, program product and method that incorporate an extensible modular communication executive for use in integrating one or more electronic devices with one another, often with substantially reduced customization overhead as compared to conventional architectures. A modular architecture is used to facilitate message-based communications in such a manner that queuing strategies, business rules and the like may be accommodated within a message-based environment in a reliable and efficient manner. Through the use of a modular architecture, application-specific software components can be assembled together to readily adapt a generic message-based system for use in a specific application. Moreover, intelligent pre-validation of messages may be implemented in such a modular architecture to permit a business rule-independent messaging infrastructure to be readily adapted to support specific business rule requirements for a particular application.

[0023] Consistent with one aspect of the invention, an active message queue architecture is utilized within a modular environment to facilitate the processing of messages sent by a sending electronic device to a receiving electronic device. The architecture includes a message buffer that is configured to receive messages, with each message associated with a message type among a plurality of message types. A plurality of software-based processor components are utilized to perform tasks associated with messages received by the message buffer, and at least one processor component is configured to initiate an action on a receiving electronic device. A plurality of software-based controller components are also utilized, with each associated with a message type among the plurality of message types, and each configured to handle a message associated with such associated message type by dynamically instantiating at least a subset of the plurality of processor components. The architecture also utilizes at least one delegator component that is associated with at least one message type among the plurality of message types and is configured to monitor the message buffer for messages associated with such associated message type. The delegator component dynamically executes a controller component associated with an associated message type in response to detecting an addition of a message associated with such associated message type to the message buffer.

[0024] Consistent with another aspect of the invention, a message buffer is integrated with a buffer services component and a plurality of message validation components, where each message validation component is associated with at least one of a message type and a receiving electronic device, with each message validation component configured to validate a message sent to the message buffer by a sending electronic device prior to addition of the message to the message buffer, and with each message validation component further configured to return a result that indicates whether the message should be accepted by the message buffer. In turn, the buffer services component processes a first message sent to the message buffer by invoking a message validation component associated with the first message and selectively adding the first message to the message buffer based upon the result returned by the invoked message validation component.

[0025] These and other advantages and features, which characterize the invention, are set forth in the claims annexed hereto and forming a further part hereof. However, for a better understanding of the invention, and of the advantages and objectives attained through its use, reference should be made to the Drawings, and to the accompanying descriptive matter, in which there is described exemplary embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] **FIG. 1** is a block diagram of a communication system incorporating a communication executive consistent with the invention.

[0027] **FIG. 2** is a block diagram of an exemplary hardware environment for the communication executive of **FIG. 1**.

[0028] **FIG. 3** is a layer diagram of the principal software components in the communication executive of **FIG. 1**.

[0029] **FIG. 4** is a flowchart illustrating the program flow of a message received routine executed by the COM service of the communication executive of **FIG. 1**.

[0030] **FIG. 5** is a flowchart illustrating the program flow of a routine executed by a delegator component in the communication executive of **FIG. 1**.

[0031] **FIG. 6** is a flowchart illustrating the program flow of a routine executed by a controller component in the communication executive of **FIG. 1**.

[0032] **FIG. 7** is a flowchart illustrating the program flow of a routine executed by a processor component in the communication executive of **FIG. 1**.

[0033] **FIG. 8** is a sequence diagram illustrating various possible execution sequences of processor components in the communication executive of **FIG. 1**.

[0034] **FIG. 9** is a flowchart illustrating the program flow of a routine executed by a message validation component in the communication executive of **FIG. 1**.

[0035] **FIG. 10** is an object diagram illustrating the principal data structures utilized in connection with managing communications with the communication executive of **FIG. 1**.

[0036] **FIG. 11** is a block diagram of one exemplary application of the communication executive of **FIG. 1**, for use in interfacing remote sensors with a central processor.

[0037] **FIG. 12** is a block diagram of another exemplary application of the communication executive of **FIG. 1**, for use in a manufacturing control system.

[0038] **FIG. 13** is a block diagram of another exemplary application of the communication executive of **FIG. 1**, for use in an inventory management system.

DETAILED DESCRIPTION

[0039] The hereinafter-described embodiments of the invention utilize a unique communication executive to provide a flexible, extensible and modular message-based interface between electronic devices, often in a manner that permits independent and disparate electronic devices, which may not have been specifically designed to communicate with one another, to communicate in a near real-time manner. As will become more apparent below, the modular and extensible architecture of the interface further often permits substantial code reuse to be employed when an interface is being adapted for use in a particular environment, thus reducing development time and increasing interface reliability.

Hardware/Software Environment

[0040] Turning to the Drawings, wherein like numbers denote like parts throughout the several views, **FIG. 1** illustrates an exemplary communication system **10** incorporating a communication executive **12** consistent with the invention. In general, the communication executive **12** is utilized to interface sending electronic devices with receiving electronic devices via a message-based protocol. A sending electronic device in this context may refer to any electronic device capable of sending or otherwise outputting information, while a receiving electronic device may refer to any electronic device capable of receiving or otherwise inputting information. Moreover, it will be appreciated that an electronic device may operate both as a sending electronic device and a receiving electronic device, and further, that a communication executive may be implemented within the same physical hardware as either or both of a sending and receiving electronic device (i.e., a communication executive may be used to manage communications within a single electronic device, and even within a single integrated circuit within an electronic device).

[0041] One useful application of a communication executive consistent with the invention, which is illustrated in **FIG. 1**, is that of interfacing various data processing devices **14** (e.g., data collection devices, data display devices, data recording devices, controller devices, etc.) with various data resources **16** (e.g., databases, data analysis tools, etc.). Moreover, as represented by the double-ended arrows interconnecting communication executive **12**, devices **14** and resources **16**, it should be appreciated that any device or resource may function from time to time as a sending electronic device, a receiving electronic device, or both.

[0042] Communication executive **12** is typically interfaced with devices **14** and resources **16** via one or more network interconnections **18**, each of which may represent practically any type of networked interconnection, including but not limited to local-area, wide-area, wireless, public networks (e.g., the Internet), and combinations thereof, as well as point-to-point serial or parallel interconnects, buses, intra-chip interconnects, and embedded interconnects. Moreover, it will be appreciated that a wide variety of additional computers and other electronic devices may be networked through the network interconnections **18** as well.

[0043] Communication executive **12** is typically implemented in software that resides and executes upon a computer or other data processing system. As shown in **FIG. 2**, for example, communication executive **12** may reside on a computer **20**, specifically within a memory **22** thereof, and be executed by one or more processors **24** interfaced with memory **22**. Processor **24** may represent one or more central processing units (e.g., microprocessors), and memory **22** may represent the random access memory (RAM) devices comprising the main storage of computer **20**, as well as any supplemental levels of memory, e.g., cache memories, non-volatile or backup memories (e.g., programmable or flash memories), read-only memories, etc. In addition, memory **22** may be considered to include memory storage physically located elsewhere in the computer **20**, e.g., any cache memory in any processor **24**, as well as any storage capacity used as a virtual memory, e.g., as stored on a mass storage device or on another computer coupled to computer **20** via a network.

[0044] Computer **20** also typically receives a number of inputs and outputs for communicating information externally. For interface with a user or operator, computer **20** may include a user interface **26**, including various user input devices (e.g., a keyboard, a mouse, a trackball, a joystick, a touchpad, and/or a microphone, among others) as well as various display devices (e.g., a CRT monitor, an LCD display panel, and/or a speaker, among others). Given that computer **20** is typically implemented as a server, user interface **26** may alternatively be implemented on a terminal or workstation interfaced with the computer, as is well known in the art.

[0045] For additional storage, computer **20** also typically includes one or more mass storage devices **28**, e.g., a floppy or other removable disk drive, a hard disk drive, a direct access storage device (DASD), an optical drive (e.g., a CD drive, a DVD drive, etc.), and/or a tape drive, among others. Furthermore, computer **20** may include an interface **30** with one or more ports to permit the communication of information between the sending and receiving devices coupled thereto, wherein each interface is dependent upon the particular network interconnection to which the computer is coupled to a particular sending or receiving device. It should be appreciated that computer **20** typically includes suitable

analog and/or digital interfaces between processor **24** and each of components **22**, **26**, **28** and **30** as is well known in the art.

[**0046**] Computer **20** generally operates under the control of an operating system (not shown separately), and executes or otherwise relies upon various computer software applications, components, programs, objects, modules, data structures, etc. (e.g., communication executive **12**, among others). Moreover, various applications, components, programs, objects, modules, etc. may also execute on one or more processors in another computer in communication with computer **20**, e.g., in a distributed or client-server computing environment, whereby the processing required to implement the functions of a computer program may be allocated to multiple computers.

[**0047**] In general, the routines executed to implement the embodiments of the invention, whether implemented as part of an operating system or a specific application, component, program, object, module or sequence of instructions will be referred to herein as "computer programs", or simply "programs". The computer programs typically comprise one or more instructions that are resident at various times in various memory and storage devices in a computer, and that, when read and executed by one or more processors in a computer, cause that computer to perform the steps necessary to execute steps or elements embodying the various aspects of the invention. Moreover, while the invention has and hereinafter will be described in the context of fully functioning computers and computer systems, those skilled in the art will appreciate that the various embodiments of the invention are capable of being distributed as a program product in a variety of forms, and that the invention applies equally regardless of the particular type of signal bearing media used to actually carry out the distribution. Examples of signal bearing media include but are not limited to recordable type media such as volatile and non-volatile memory devices, floppy and other removable disks, hard disk drives, magnetic tape, optical disks (e.g., CD-ROM's, DVD's, etc.), among others, and transmission type media such as digital and analog communication links.

[**0048**] In addition, various programs described hereinafter may be identified based upon the application for which they are implemented in a specific embodiment of the invention. However, it should be appreciated that any particular program nomenclature that follows is used merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature.

[**0049**] Those skilled in the art will recognize that the exemplary environment illustrated in FIGS. **1-2** is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware and/or software environments may be used without departing from the scope of the invention.

Network-Based Communication Executive

[**0050**] One implementation of communication executive **12** of FIGS. **1-2** is that of interfacing various electronic devices coupled over a communication network such as the Internet, a Local-Area Network, a Wide-Area Network, and combinations thereof, using a conventional network-based protocol such as TCP/IP. It is this specific implementation that will be focused upon hereinafter, it being understood that the present invention is not limited solely for use in such an implementation.

[**0051**] Overview of Principal Software Components

[**0052**] **FIG. 3** illustrates the principal software components in communication executive **12**. Several layers of software are defined at **40-54**, and are used to provide a flexible, extensible and modular communications service for linking electronic devices to one another in a structured manner.

[**0053**] External device layer **40** represents the source and destination devices that utilize the communication executive. It will be appreciated that each external device will typically include its own queuing strategy and may require various business rules to control the messages sent by or received by such devices.

[**0054**] Network services layer **42** represents the network protocol services required to interface the communication executive with the particular type of networks over which the external devices are accessible. COM service layer **44**, which represents a buffer services component, provides an external interface to a message repository or database, and in the illustrated embodiment, is implemented as a Visual Basic COM object that acts as the liaison between the message repository and any system that wishes to enter tasks into or retrieve information from the repository. Typically, the COM service layer is called by the network services layer **42**, or other non-network based systems directly, e.g., via DCOM calls, in a manner well known in the art.

[**0055**] It will be appreciated that the network and COM service layers **42**, **44** may require customization for use in different environments. For example, depending upon the type of networks over which various devices are coupled to the communication executive, the network services may require differing functionality. For example, where external devices are coupled over the Internet or another TCP/IP network, the network services may function as a listening device, capable of receiving TCP/IP packets and forwarding packets in a predetermined format to the communication executive as required. For other forms of network protocols, or in situations where listening or monitoring is not required, different network services functionality may be required, or the network services may be omitted altogether. Likewise, the COM service into the message repository may vary in other applications. In the illustrated embodiment, the COM service is instantiated whenever data is received, and the COM service is configured to dynamically generate requests from received data. Given the variable types of devices that may be coupled to a communication executive, however, the COM service will differ for various applications.

[**0056**] Next, as will be discussed in greater detail below, one or more message validation (MV) components **46** may also be provided in communication executive **12**. Each MV component performs pre-validation of received messages for particular types of messages and/or for messages directed toward particular receivers. The MV components typically implement desired business rules, and are used to validate a message using artificial intelligent techniques before the message is added to a message repository. In this manner high-level message processing may occur prior to the message being accepted onto a message repository. In other embodiments, however, MV components may not be used.

[0057] The MV components are layered on top of a message repository, also referred to herein as a cube **48**, which represents the database or buffer within which messages are stored. When coupled with one or more delegators **50**, message repository **48** forms an active queue that actively manages messages placed on the buffer by the COM service.

[0058] Each delegator **50** functions as a monitor task that actively monitors for messages of a particular type. As will be discussed in greater detail below, a delegator, in response to detecting a new message for which it is responsible, dynamically spawns (or otherwise executes) one or more controller components **52** to handle such messages. Typically, a delegator will call a controller for each message type for which the delegator is responsible. In the illustrated embodiment, each delegator is implemented as a Visual Basic program such as an ActiveX executable, although other software implementations may be used in the alternative.

[0059] As discussed above, each controller component **52** is dynamically instantiated or spawned by a delegator to handle a particular message type. In other embodiments, controller components may remain resident at all times, and as a result, may be executed by a delegator on an as-needed basis. Instantiation or spawning in addition to execution is therefore optional in some embodiments.

[0060] In the illustrated implementation, each controller represents a logical layer split into two physical layers. A controller is capable of handling complex chain processing, with a CBO layer implemented as a Visual Basic program, e.g., an ActiveX executable, that calls an SBO controller to process the requests from the cube for a particular message type. The SBO layer may be implemented as a Visual Basic program such as a COM+ ActiveX DLL that calls one or more processor components **54** dedicated to handling a specific message type to process the relevant messages in the message repository. Through the use of a dual-layer implementation, a delegator may be allowed to process requests of different types without having to wait for any of the requests to complete. However, it should be appreciated that other software implementations may be used for a controller consistent with the invention.

[0061] Each processor component **54** is implemented in the illustrated embodiment as a Visual Basic COM object that contains the business logic for a particular system using the communication executive as its queuing engine. The processor components are typically resident outside of the communication executive, and are used to interact directly with destination devices to actually process messages in the manner appropriate for a particular device. The processor components read messages from the message repository and interface with a destination device either directly or by adding new out-bound messages to the message repository via the CBO layer of the associated controller. As with the other components in the communication executive, however, other software implementations may be used for processor consistent with the invention. In general, a processor component can perform any number of tasks associated with a message, including in many instances initiating some action on a destination device.

[0062] **FIG. 3** also illustrates an optional system analyzer **56**, which may represent one or more self-diagnosis routines that constantly monitor the health of the communication executive. Typically, when an analyzer detects a problem, an entry may be logged in, and depending upon the severity of the error, a message may be sent to an administrator. In this regard, the system analyzer functions in a similar manner to other network monitoring products.

[0063] In use, a device wishing to send a message via the communication executive to another device in the system will initiate a function call to the COM service associated with that device, which, if the message is acceptable, will place the message on the message repository. Then, a delegator associated with the particular type of message will dynamically spawn one or more controllers to handle the message. The appropriate controller will in turn call one or more processor components to actively handle the task. The result of handling a message may also incorporate sending a response message or initiating another message for another device coupled within the system. The details of this overall program flow are illustrated in greater detail in FIGS. **4-9** below.

[0064] Message Reception

[0065] **FIG. 4** illustrates a message received routine **60**, executed for example by COM service **44** whenever a message needs to be placed on the message repository. Routine **60** begins in block **62** by determining whether a message validation component needs to be called for the message. As discussed above, typically message validation components, if used, are associated with particular message types, and are used to pre-validate a message before the message is placed on the message repository, or cube. Therefore, if such a component exists for the particular message type with which the message is associated, block **62** passes control to block **64** to call the appropriate message validation component. The message validation component typically has the ability to either accept, reject or modify an incoming message. If the message is rejected, control passes to block **66** to terminate routine **60** and return an error, indicating to the calling component that the message was not accepted. Otherwise, control passes to block **68** to enqueue the message onto the cube, e.g., by storing the message in the message repository. Once the message has been placed on the repository, routine **60** is then complete.

[0066] Returning to block **62**, if no message validation component is appropriate for the particular message that has been received, block **62** passes control directly to block **68** to enqueue the message onto the cube.

[0067] Delegator Component/Active Queue

[0068] Once a message has been placed on the message repository, an active monitoring process associated with the repository is utilized to automatically handle the message by dynamically instantiating one or more controller components associated with the message type for that message. **FIG. 5**, for example, illustrates a delegator routine **70** executed by one of the delegator components resident in the system. As discussed above, a delegator component may be implemented as a Visual Basic ActiveX executable program, among other implementations. Once initiated, a delegator will first connect to the message repository and create a list of available delegator components, as shown in block **72**. Once the initial connections have been established, the delegator is ready to receive events, and to monitor for new

messages placed on the cube. Thus, the delegator waits on a wait block **74**, for various types of events that may be directed to the delegator. In addition, while waiting for events, the delegator also periodically monitors the cube for the receipt of a new message. In other embodiments, events may be handled in a different thread from message monitoring.

[**0069**] In some implementations, it may be desirable to selectively disable and enable a delegator, and as such, a delegator may be disabled or enabled in response to suitable events, as shown in blocks **76** and **78**. Likewise, it may be desirable to shut down a delegator, as shown in block **80**.

[**0070**] As discussed above, block **74** also monitors for the placement of a new message on the cube. In response to receipt of a new message, block **82** is called to determine whether the delegator is currently enabled. If not, the message is ignored, and control returns to block **74**. If the delegator is enabled, however, block **82** passes control to block **84** to retrieve the message data from the cube. Block **86** then determines whether any other delegator is currently handling the message, e.g., by determining whether a flag or indicator set in the message record indicates that the message is associated with another delegator. If so, no further processing by the delegator occurs, and control returns to block **74**. Otherwise, control passes to block **88** to associate the message with a delegator by changing a flag or other indicator in the message record.

[**0071**] Next, block **90** determines what controller is required for processing the message, typically based upon the type of message as found in the message record, as well as a known mapping between controllers and messages defined within the system. Block **92** then dynamically instantiates, or spawns, such controller, which also incorporates execution of the spawned controller. Control then returns to block **74** to process other events and messages.

[**0072**] It will be appreciated that additional information may be presented to an administrator by a delegator component consistent with the invention. For example, information such as total cycles started and total cycles completed may show an administrator the number of times a delegator has searched the cube for messages, and the number of times the delegator has completed the search. In addition, a cycle time, representing the period during which the delegator periodically checks the cube, may also be configurable and displayed to a user. Status information may also be displayed illustrating whether the delegator is currently enabled or disabled, as well as indicating when messages are received and when controllers are spawned in response to reception of such messages. A status window displayed to an administrator may also include various user interface controls such as START and STOP buttons that selectively enable or disable the delegator, as well as an END button that shuts down the delegator in the manner described above.

[**0073**] Controller Component

[**0074**] **FIG. 6** next illustrates a controller routine **100** executed by a controller component consistent with the invention. As described above, the controller component is typically divided into two physical layers, a CBO controller layer and an SBO controller layer, with the CBO controller layer configured as a single-use (class parameter) ActiveX executable, and the SBO controller layer being implemented as a COM+ ActiveX DLL Visual Basic program.

[**0075**] In the illustrated implementation, the CBO controller layer includes a hidden form with a timer control that is initially inactive when instantiated by a delegator component. A method, e.g., a start controller method, may be exposed by the controller, and include a controller ID parameter, delegator ID parameter, a delegator pointer and error information. The method, when called by a delegator, stores the parameters in private variables, loads the hidden form, and enables the time control on the form. Then, the method ends and returns control to the calling delegator, which frees the delegator to process messages of a different system ID/message type, while the controller proceeds with processing of messages. The timer control may be set to fire some delay after being enabled, e.g., one second. Once the time control fires, an instance of an SBO controller is created and messages are processed as described below. Once the process messages method has completed, a DONE method exposed by the delegator may be called to return error and status information to the delegator.

[**0076**] As shown in **FIG. 6, a** controller routine **100** generally begins in block **102** by connecting to the message repository and obtaining a list of messages that need to be processed. Next, block **104** obtains a list of processors that are to be launched, including the desired order. Block **106** then initiates a FOR loop to process each of the processors necessary to be launched. For each such processor, control passes to block **108** to determine whether the processors needs a transaction.

[**0077**] In the illustrated embodiment, processors may be grouped into transactions, with transactional groups defined within the message repository. Processors may be defined within the same transactional group based upon processing order and whether or not each processor is batch or single record. By being placed in a transactional group, multiple processors can apply multiple changes to a database or multiple databases and have the changes committed or rolled back as a group.

[**0078**] If a processor does not need a transaction, block **108** simply passes control to block **110** to launch the appropriate processor, e.g., by instantiating the processor or making a function call to the processor as appropriate. Control then passes to block **112** to determine whether the last processor has been reached, whereby control returns to block **106** to process additional processors as appropriate.

[**0079**] On the other hand, if a processor requires a transaction, block **108** passes control to block **114** to determine whether the message is part of a previous transaction. If not, control passes to block **116** to begin the transaction, and then to block **118** to launch the appropriate processor. Otherwise, if the message is part of a previous transaction, block **114** passes control directly to block **118**. Once the appropriate processor has been launched, control passes to block **120** to determine whether the next processor is part of the same transaction. If not, control passes to block **122** to commit the transaction. Regardless of whether the next processor is part of the same transaction, control then passes to block **112** to determine whether the last processor has been launched. If not, control returns to block **106**. Otherwise, control passes to block **124** to commit any open transactions, and then to block **126** to update the statistics associated with the message. Upon completion of block **126**, routine **100** is complete.

[0080] In the illustrated embodiment, the determination of what processors are to be launched for a particular type of message is made by accessing a database such as the message repository, within which such information is stored. The settings may include information as to which processors to call and in what order, as well as whether each of these processor is a batch or single record processor. In many instances, the information is stored in connection with data records that map specific processors to specific controllers, e.g,. in the tblControllerProcessors table 196 discussed below in connection with **FIG. 10**.

[0081] Typically, each processor has one method, e.g., an execute method, that is called by a controller to invoke the functionality in the processor. The execute method may include a record ID field, as well as option delegator ID, system ID, message type, text data, binary data, error code data, etc., with the record ID used to identify the function call as either a single record call, where the ID of the particular message record is passed, or a batch-type call, where a unique token, e.g., "−1" is used in the record ID field. The error information and text and binary data may be provided to allow a controller to pass data from a previously-called processor to a next processor. Moreover, an execute method may be configured to return a status to the controller, indicating whether the execution was successful. If an unsuccessful execution occurs, the error information may be processed by the controller as appropriate, i.e., by rolling back the transaction, aborting processing of the message, and/or updating the message status. It may also be desirable to keep track in the controller of the time it takes to process a message, and use this information to store statistical information regarding the operation of the controller. The time to process may also be utilized to trigger a watchdog timer should a processor not process a message in an appropriate period of time.

[0082] Processor Component

[0083] **FIG. 7** illustrates a processor routine **130** executed by generic processor component consistent with the invention. In general, routine **130** represents the execute method exposed by the respective processor. Once invoked, block **132** is called to read the message record for the relevant data, and then to block **134** to perform processing on the message as required.. Block **136** then determines whether the processor is being executed in batch mode (i.e., where a batch mode is indicated in the method call), and if so, control passes to block **138** to determine whether the last message has been processed.

[0084] As described above, when in batch mode, a unique token is applied in the function call that indicates to the processor component that all messages of a particular message type should be processed. Thus, block **132**, when in batch mode, initially reads the first message matching the particular message type handled by the associated controller. Otherwise, in single record mode, the message record identified in the method call is read. Subsequent loops through block **132** when in batch mode will call additional unprocessed messages maintained in the message repository.

[0085] Returning to block **136**, if not in batch mode, block **136** passes control to block **140** to return appropriate success or failure information to the calling controller, as well as provide state information (if applicable) to the controller. Also, as shown in block **138**, if the last message in a batch mode has been processed, block **138** passes control to block **140**.

[0086] Once block **140** is complete, routine **130** terminates.

[0087] It will be appreciated that the processing performed by a processor component may vary greatly depending upon the particular type of device being managed by that processor. For example, a processor may be configured to send a message to its associated external device to perform tasks on that device, and in a format that is understood by the device. Moreover, a processor may respond to a particular message by returning status information or requested information, e.g., by modifying the message record as appropriate. A processor may also be configured to generate new messages, e.g., response messages, which are placed on the message repository and processed in a like manner. Other functions, e.g., updating the database, sending an email, posting data to a web page, interfacing with another system, or practically any other function capable of being performed by a computer, may also be performed by a processor consistent with the invention.

[0088] The configuration of the controller and processor components as described above in connection with **FIGS. 6 and 7** presupposes that processors generally execute sequentially and within the same thread. However, in other embodiments, a wider variety of execution sequences may be supported, thus providing an extremely flexible and modular architecture by which custom functions may be generated from pre-existing processor components to support a wide variety of functionality while minimizing the amount of new program code that needs to be generated to implement a communication executive in a different environment. For example, **FIG. 8** illustrates an exemplary controller **150** and various function calls to a number of processors **152** in response to the reception of a particular type of message. As shown by processors A, B and C, a controller may be capable of spawning multiple processors that execute concurrently in different threads. Moreover, as illustrated by processor D, controller **150** may also spawn processors sequentially within the same thread. Furthermore, as illustrated by processors E and F, not only can a controller spawn a processor, but one processor may spawn one or more additional processors that are executed in the same or different threads, and which may execute sequentially or concurrently in those different threads. In general, the modular architecture by the herein-described communication executive provides substantial flexibility and maximizes the ability to reuse code from generic or pre-existing environments.

[0089] Message Validation Component

[0090] **FIG. 9** next illustrates an exemplary message validation component routine **160** executed by a message validation component consistent with the invention. As described above, a message validation component may be dynamically instantiated in response to reception of a message of a particular type and/or destined for a particular receiving device, and preferably before the message is placed on the message repository. The purpose of a message validation component is to perform high-level message validation prior to placement of the message on the repository, so as to minimize the amount of custom high-level

processing that must occur via the controllers and processors, and thus limit the degree of customization required in such components.

[0091] Routine **160** begins in block **162** by reading the message data passed from the COM service. Block **164** then analyzes the message data according to business rules that are encoded into the validation component. The message validation component operates generally as a plug-in associated with a message type and/or a particular message receiver, and is configured to analyze, validate, modify or perform whatever tasks are necessary to properly format the message for processing by the communication executive. In general, a message validation component has the capability of accepting a message, modifying a message, or rejecting a message and returning any errors to the calling service. An advantage of message validation is the ability to provide real-time validation of messages so a requester does not need to check back to see if the message was accepted or rejected.

[0092] Data analysis in block **164** generates a result that indicates whether the message should be accepted, rejected or accepted with changes. As shown in block **166**, if the message is to be rejected, control passes to block **168** to return "FALSE" to the COM service, indicating that the message has been rejected. Routine **160** is then complete.

[0093] As shown in block **170**, if the message is accepted without changes, control passes to block **172** to return "TRUE" to the COM service, and terminate routine **160**.

[0094] As shown in block **174**, if modifications are required prior to accepting the message, such modifications are performed, then control passes to block **172** to return "TRUE"; to the COM service and terminate routine **160**.

[0095] Therefore, it can be seen through appropriate analysis according to high-level business rules, message validation can be performed with minimal customization required to the overall communication executive. This modular and extensible approach therefore facilitates incorporating business rules into a communication system with minimal customization.

[0096] Exemplary Data Structures

[0097] **FIG. 10** generally illustrates one exemplary implementation of the data structures maintained within a message repository **180** consistent with the invention. Messages are maintained in message record table 182. Binary and text data are respectively associated with a particular message using the tables shown at 184 and 186, respectively. Message records are also associated with particular "systems" via a table 188 that maintains the valid system ID's and message types. Controller records are stored in a controller table 190 with a mapping record **192** linking a controller to a delegator defined in a delegator record **194**. Likewise, each controller is linked to processors via a table 196, with processors defined by processor records in table 198. Statistical data is maintained in a statistics table 200, with error data maintained in an error table 202, linked to binary and text data in tables 204 and 206.

[0098] Descriptions of the exemplary fields within each of tables 182-206 of **FIG. 10** are provided below:

TABLE 182

is the main queue (the Cube) for all tasks:

Field Name	Field Type	Description
MsgID	Auto (Number)	The ID generated for the Cube record.
FromID	VarChar2	The system ID that generated the message.
ToID	VarChar2	The target system ID for the message.
MsgType	VarChar2	The target message type.
DateTime	Date	The date and time stamp when the record was entered into the Cube.
Status	Number	The current status of the message. Valid values are: 0 (New), 1 (In Progress), 2 (Complete), or 3 (Error).
Misc1	VarChar2	Free form text field. Contains any data necessary for the message to be processed.
Misc2	VarChar2	Free form text field. Contains any data necessary for the message to be processed.
Misc3	VarChar2	Free form text field. Contains any data necessary for the message to be processed.
Misc4	VarChar2	Free form text field. Contains any data necessary for the message to be processed.
Misc5	VarChar2	Free form text field. Contains any data necessary for the message to be processed.
DelegatorID	Number	The ID of the Delegator that is currently working on this task.
Priority	Number	Priority of this task.
ErrorCode	Number	The ID of the error code (if the task failed).
ErrorMessage	VarChar2	The error message generated (if the task failed).

[0099] 1. tblCube Table

TABLE 194

contains all the Delegators and their settings:

Field Name	Field Type	Description
DelegatorID	Auto (Number)	ID of the Delegator.
DelegatorDescription	VarChar2	Description of the Delegator.
CycleTime	Number	Number of seconds this Delegator will wait before looking into the Cube for records for each system.
OkToRun	Number	If set to Yes, then this Delegator has not been shut down.

[0100] 2. tblDelegators

TABLE 190

contains all the Controllers and their settings:

Field Name	Field Type	Description
ControllerID	Auto (Number)	ID of the Controller.
SystemID	VarChar2	The System ID that this Controller is responsible for.
MsgType	VarChar2	The type of message that this Controller is responsible for.

[0101] 3. tblController

TABLE 198

contains all the Processors and their settings:

Field Name	Field Type	Description
ProcessorID	Auto (Number)	ID of the Processor.
ProcessorClass	VarChar2	String used in the VB program to instantiate the COM object (i.e. 'prjProcessor1.clsMain').
BatchProcessing	Number	This indicates whether the Controller will call the Processor for each message (single record), or once for all messages (batch).
IsTransactional	Number	If Yes, transactional processes have been implemented and the system should check Object Context values when processing.

[0102] 4. tblProcessors

TABLE 192

links a Delegator with its Controllers:

Field Name	Field Type	Description
DelegatorID	Number	The ID of the Delegator that will call this Controller.
ControllerID	Number	The ID of the Controller that this Delegator will call.
ControllerOrder	Number	The order in which this Delegator will call this Controller

[0103] 5. tblDelegatorControllers

TABLE 196

links a Controller with its Processor:

Field Name	Field Type	Description
ControllerID	Number	The ID of the Controller that will call this Processor.
ProcessorID	Number	The ID of the Processor that this Controller will call.
ProcessOrder	Number	The order in which this Controller will call this Processor.

[0104] 6. tblControllerProcessors

TABLE 202

is updated with any errors that occur during the processing of a message:

Field Name	Field Type	Description
ID	Auto (Number)	ID of the error record.
QueueID	Number	ID of the task from the Cube.
From ID	VarChar2	The system ID that generated the task.
ToID	VarChar2	The system ID that received the task.
MsgType	VarChar2	The message type.
ErrorCode	Number	The error code generated during the processing of this message.
ErrorMessage	VarChar2	The error message generated during the processing of this message.
DateTimeEntered	Date	The Date/Time stamp when the message was entered.
DateTimeError	Date	The Date/Time stamp when the error occurred.

TABLE 202-continued

is updated with any errors that occur during the processing of a message:

Field Name	Field Type	Description
Misc1	VarChar2	Value of the corresponding field from the Cube.
Misc2	VarChar2	Value of the corresponding field from the Cube.
Misc3	VarChar2	Value of the corresponding field from the Cube.
Misc4	VarChar2	Value of the corresponding field from the Cube.
Misc5	VarChar2	Value of the corresponding field from the Cube.
ErrorProcessor	VarChar2	The Processor (description) where the error occurred.
Priority		The priority of the message from the Cube.
DelegatorID	Number	The ID of the Delegator that was processing this message.

[0105] 7. tblCube Error

TABLE 188

includes the valid System ID's and message types:

Field Name	Field Type	Description
SystemID	Varchar2	The name of the system.
MsgType	VarChar2	The message type.
AIProcessorClass	VarChar2	The string used in the CreateObject statement for the MV processor (optional).
MaxTimeThreshold	Number	The maximum length of time in seconds that a message of this type should take to process. A value of −1 indicates no threshold.

[0106] 8. tblSystems

TABLE 200

includes statistical data:

Field Name	Field Type	Description
StatisticalDate	Date	The date when the statistics were gathered.
SystemID	VarChar2	The ID of the System for these statistics.
MsgType	VarChar2	The message type.
LastSuccess	Date	The last date/time when a message was successful.
LastFailure	Date	The last date/time when a message failed.
TotalAttempts	Number	Number of attempts.
TotalFailed	Number	Number of failures.
CurrentDate	Number	Current date.

[0107] 9. tblStatistics

TABLE 218

includes all long text data for a message:

Field Name	Field Type	Description
MsgID	Number	ID of the message.
TextData	Long	Field that contains all text data that will be necessary to complete the task.

[0108] 10. tblCubeText

TABLE 188

includes all binary data for a message:

Field Name	Field Type	Description
MsgID	Number	ID of the message.
BinaryData	Long Raw	Field that contains all binary data that will be necessary to complete the task.

[0109] 11. tblCubeBinary

TABLE 206

includes all long text for an error record:

Field Name	Field Type	Description
ID	Number	ID of the message.
TextData	Long	Field that contains all text data that will be necessary to complete the task.

[0110] 12. tblCubeErrorText

TABLE 204

includes all binary data for an error record:

Field Name	Field Type	Description
ID	Number	ID of the message.
BinaryData	Long Raw	Field that contains all binary data that will be necessary to complete the task.

[0111] 13. tblCubeErrorBinary

[0112] In the illustrated implementation, the system ID refers to the type of communication system utilized in the communication executive. As will be appreciated by one of ordinary skill in the art having the benefit of the instant disclosure, multiple systems may be interfaced through the same communication executive. In situations where only a single system is being interfaced in the manner described herein, the system ID-related fields may be omitted from the various tables.

[0113] Moreover, it will be appreciated that the message types defined for a particular system will vary from system to system. Message types may be distinguished based upon the type of receiving device, the type of sending device, the type of resource being utilized, the type of transaction (e.g., based upon logical distinctions between activities performed with the system), etc.

[0114] Exemplary Message Repository Stored Procedures

[0115] While other specific sets of procedures may be envisioned in different environments, one exemplary set of message repository stored procedures that may be utilized in connection with the above-described data structures are listed below. Such methods are made available by the message repository. In addition, it may be desirable to permit processor components to invoke these methods if data is required from a message, or if data needs to be written to a message.

[0116] In implementations where the message repository is implemented using an SQL-compatible database, it is envisioned that the below procedures may be created using SQL statements, the use of which is well known in the art.

[0117] The exemplary procedures are as follows:

sp_SendMessage	Writes a record from the Cube.
sp_RemoveMessage	Deletes a record from the Cube.
sp_PurgeAll	Purges all records from the Cube with a given "to name" and message type.
sp_UpdateStatus	Updates the status of a record to a supplied value.
sp_InsertError	Inserts an error into the Error Log.
sp_AddStatistics	Adds a new statistics record to the tblStatistics table.
sp_AutoAddStatistics	Adds new statistical record for each calendar day. May be executed through a job that runs once every day at 12:00 a.m.
sp_UpdateStatistics	Updates statistics from the tblStatistics table.
sp_UpdateMsgDelegatorID	Updates the Delegator ID field in a Cube record with the ID of the Delegator that is processing the message.
sp_UpdateDelegatorStatus	Updates the status of a Delegator. Read to determine if a Delegator is free to look for messages in the Cube.
sp_RemoveOldStatistics	Removes statistics from the tblStatistics table that are older than 30 days. This procedure may be scheduled to run automatically each day.
sp_Update Delegator Status	Updates the OkToRun field for the Delegator.

[0118] It will be appreciated that the implementation of both the above-described data structures and the stored procedures would be well within the ability of one of ordinary skill in the art having the benefit of the instant disclosure.

[0119] Exemplary COM Service Methods

[0120] The COM services may expose the following methods to calling systems to execute. The COM service object may be stateless—that is, all data needed for a method to function may be passed in as parameters, and all variable within the object may be declared as local variables:

ReadAllMessages(ToID, MsgType, OUT_Error_Code, OUT_Error_Description, OUT_Messages)

[0121] Returns an XML Stream of Record ID's that are currently in the Cube that match the ToID and MsgType parameters. Returns TRUE if successful or FALSE if unsuccessful. If the method returns FALSE, the Error_Code and Error_Description contain the error code and error message that occurred. The Messages parameter may be an XML stream with the following layout:

```
<Messages>
    <MsgID></MsgID>
    <MsgID></MsgID>
    ...
</Messages>
```

[0122] This XML stream can then be parsed and the ReadMsgData method can be called to obtain individual record information from the Cube.

SendMsg (FromID, ToID, MsgType, Priority, OUT_MsgID, OUT_Error_Code, OUT_Error_Descrip-

tion, Optional Misc1, Optional Misc2, Optional Misc3, Optional Misc4, Optional Misc5, Optional TextDataIn, Optional BinaryDataIn)

[0123] Adds a record to the Cube. Returns TRUE if successful, FALSE if unsuccessful. If the return value is FALSE, then Error_Code and Error_Description may contain the error code and the error message that occurred while attempting to add this record. The Send method may automatically call the Execute method for the MV component associated with this SystemID and MessageType (if one exists). If the method returns TRUE, then the MsgID will contain the ID of the message from the Cube that can be used in conjunction with the Delete, CheckStatus, and ReadMsgData methods.

[0124] Regarding the Send method, regardless of whether or not the Delegator for that system has been disabled or closed, the COM Service will still typically enter records into the Cube.

DeleteMsg (MsgID, OUT_Error_Code, OUT_Error_Description)

[0125] Removes a message from the Cube based on the RecordID. Returns TRUE if successful or FALSE if unsuccessful. If the Delete method was unsuccessful, the Error_Code and Error_Description parameters may contain the actual error code and error message that occurred.

Purge (ToID, MsgType, OUT_Error Code, OUT_Error_Description)

[0126] Removes all messages from the Cube that match the ToID and MsgType. Returns TRUE if successful and FALSE if unsuccessful. If the Purge method was unsuccessful, the Error_Code and Error_Description parameters may contain the actual error code and error message that occurred.

CheckMsgStatus (MsgID, OUT_MsgStatus, OUT_Error_Code, OUT_Error_Description)

[0127] Checks the status of a message within the Cube based on the MsgID. Status will contain the current status of the Cube message (see Status in the tblCube table above). Returns TRUE if successful and FALSE if unsuccessful. If the CheckStatus method was unsuccessful, the Error_Code and Error_Description parameters may contain the actual error code and error message that occurred.

ReadMsgData (MsgID, OUT_FromID, OUT_ToID, OUT_MsgType, OUT_Priority, OUT_Misc1, OUT_Misc2, OUT_Misc3, OUT_Misc4, OUT_Misc5, OUT_TextDataIn, OUT_BinaryDataIn, OUT_DelegatorID, OUT_Error_Code, OUT_Error_Description)

[0128] Returns all the values from the Cube fields based on the MsgID into the parameters above. Also returns TRUE or FALSE. If the method returns TRUE, then the Error_Code parameter may contain the error code and the Error_Description parameter may contain the error message.

CheckDelegatorStatus (DelegatorID, OUT_DelegatorStaus, OUT_Error_Code, OUT_Error_Description)

[0129] Checks the status of the Delegator based on the Delegator ID passed in. The OkToRun value will be returned in the OUT_DelegatorStatus field. This value may be a Boolean value.

DeleteMsg (MsgID, OUT_Error_Code, OUT_Error_Description)

[0130] Deletes a message based on the MsgID value. OUT_Error_Code and OUT_Error_Description may contain the error number and error description if any error occurs.

GenerateErrorRecord (FromID, ToID, MsgType, Priority, MsgID, MsgError_Code, MsgError_Description, ErrorDate, Processor, DelegatorID, OUT_Error_Code, OUT_Error_Description, Optional Misc1, Optional Misc2, Optional Misc3, Optional Misc4, Optional Misc5, Optional TextDataIn, Optional BinaryDataIn)

[0131] Generates a new error message.

GetAllControllerProcessors (ControllerID, OUT_Processors, OUT_Error Code, OUT_Error_Description)

[0132] Returns an XML stream of all the Processors based on the Controller ID. The XML may have the following format:

```
<Processors>
    <ProcessorID></ProcessorID>
    <ProcessorOrder></ProcessorOrder>
</Processor>
```

GetAllDelegatorControllers (DelegatorID, OUT_Controllers, OUT_Error Code, OUT_Error_Description)

[0133] Returns an XML stream of all the Controllers based on the Delegator ID. The XML may have the following format:

```
<Controllers>
    <ControllerID></ControllerID>
    <ControllerOrder></ControllerOrder>
</Controllers>
```

GetAllDelegators (OUT_Error Code As Variant, OUT_Error_Description, OUT_Delegators)

[0134] Returns an XML stream of all the Delegators. The XML may have the following format:

```
<Delegators>
    <DelegatorID></DelegatorID>
</Delegators>
```

GetControllerInfo (ControllerID, OUT_SystemID, OUT_MsgType, OUT_Error_Code, OUT_Error Description)

[0135] Returns the settings for a Controller based on the Controller ID passed in.

GetDelegatorInformation (DelegatorID, OUT_DelegatorDescription, OUT_CycleTime, OUT_DelegatorStatus, OUT_Error Code, OUT_Error_Description)

[0136] Returns all of the Delegator information based on the Delegator ID passed in.

GetProcessorInfo (ProcessorID, OUT_ProcessorClass, OUT_BatchProcessing,OUT_IsTransactional, OUT_Error_Code, OUT_Error_Description)

[0137] Returns Processor information based on the Processor ID passed in.

GetSystemInfo (SystemID, MsgType, OUT_AIClass, OUT_MaxTimeThreshold, OUT_Error_Code, OUT_Error_Description)

[0138] Returns information from tblSystems based on the System ID.

UpdateDelegatorStatus (DelegatorID, NewStatus, OUT_Error_Code, OUT_Error_Description)

[0139] Updates the Delegator OkToRun status with the NewStatus Boolean value.

UpdateMsgDelegatorStatus (MsgID, DelegatorID, OUT_Error_Code, OUT_Error_Description)

[0140] Updates the message in the Cube with the Delegator ID passed in.

UpdateMsgStatus (MsgID, NewStatus, OUT_Error_Code, OUT_Error_Description)

[0141] Updates the status of the message to the NewStatus value passed in.

UpdateStatistics (SystemID, MsgType, ProcessedOK, StatDate, OUT_Error_Code, OUT_Error_Description)

[0142] Updates the current statistic record for the System ID and Message Type. ProcessedOK may be a Boolean value.

[0143] Exemplary Applications

[0144] The above-described communication executive has a wide variety of uses in various applications, particularly where it is desirable to interconnect electronic devices having different queuing strategies and/or business rules.

[0145] For example, **FIG. 11** illustrates a remote sensing environment **220** that utilizes a central processor **221** including a communication executive **222** that interfaces a plurality of remote sensing stations, or sensors, **224** with a data repository **225** and analysis engine **226**. Additional details regarding such a system are described, for example, in the aforementioned related U.S. Provisional Application No. 60/307,348, which is incorporated by reference herein.

[0146] In this remote sensing environment, centralized analysis functionality may be used to permit, for example, independent entities to contract for analysis services to be provided by the central processor, needing only access to an NIR sensor and appropriate networking capabilities. In such a system, the relatively complex processing required to perform quantitative analysis can be maintained in a central location, thus reducing the complexity and cost of the remote stations. Moreover, any required customization of remote stations can be minimized. Further, centralized analysis allows for greater control over the models used in the analysis, as well as facilitates updates or improvements to the models since they are typically not distributed to the remote stations.

[0147] Each remote sensor **224** may include, for example one or more near infrared (NIR) instruments **228** interfaced with a client-side computer, or user interface, **230**, e.g., a laptop or other single-user computer. In this exemplary application, the NIR instruments **228** are utilized to generate spectral data for a material for the purpose of performing quantitative and/or identification analysis on materials or compositions located at remote locations. The spectral data generated by the remote sensors is then transmitted either in raw or pre-processed form to a central processor including an analysis engine that analyzes the spectral data using chemometrics or other multivariate methods to determine the quantitative properties of a particular material or composition. The analysis engine may also return the processed results to the remote station from which the data was generated.

[0148] Each remote sensor may utilize, for example, a Matrix-F NIR instrument available from Bruker Optics Inc. A laptop or other web-enabled computer may then be used to provide user input and display capability for communicating with a central web server. In some embodiments, a client computer may not be required, e.g., if the instrument itself incorporates suitable networking and display capabilities. Other NIR instruments or sensing systems may be used in the alternative as well.

[0149] To interface central processor **221** with the remote sensors **224**, it is envisioned that the remote sensors would be coupled to the communication executive **222** via a network such as the Internet **232**. Through the use of conventional Internet-based based protocols, the spectral data (as well as other measurement data such as input data received from an operator) and the processed measurement results may be transmitted via HTML-compatible forms processed by a web server **234**, which is interposed between the sensors and the communication executive **222**. The web server may incorporate practically any conventional web server technology, e.g., Active Server Page (ASP) services.

[0150] Analysis engine **226** includes an analysis controller **240** coupled to a model engine **242**, which relies on stored models **244** in data repository **225** to apply chemometric analysis to spectral data generated by the remote sensors. An analysis database **246** is also provided in data repository **225** to store the spectral data and processed results. Databases **244** and **246** may be integrated into the same database management system, or may be maintained in separate such systems.

[0151] Communication executive **222** includes a cube **250** that is accessed via a COM Service **252** instantiation responsive to requests received by web server **234**. A delegator **254** operates on messages received by cube **250**, resulting in the dynamic instantiation of a controller **256** responsive to an appropriate message. Controller **256** has access to one or more processors **258** functioning as request handlers for the central processor. Each processor **258** is coupled to receive and transmit data from and to analysis database **246**, as well as to forward requests in an appropriate format to analysis controller **240**.

[0152] Among other benefits, the herein-described system of **FIG. 11** provides the capability for near real-time processing and scalability to permit a large number of remote sensors to access centralized models for the purpose of processing spectral data and returning measurement results to the remote sensors. As a result, the local processing required at each remote sensor can be minimized.

[0153] Moreover, more detailed and comprehensive models may be maintained in the central processor for each remote sensor, and updated over time to improve the precision and accuracy of the results generated by the central processor.

[0154] Given the distributed nature or the aforementioned system and the ability to couple the remote sensors to the central processor over a public network such as the Internet, the number of remote sensors requesting test results will typically be unknown at any given point in time, so the communication executive must be capable of handling a large number of requests at a time. Through the use of the active queue and dynamic instantiation capabilities of the herein-described communication executive, varied numbers of incoming requests may be adequately handled despite any variances in the queuing strategies of the remote sensors and the central processor.

[0155] The herein-described communication executive provides first-in-first-out queuing, as well as automated message detection to enable near real-time processing. Moreover, the pipeline-processing framework facilitates extension of the message processing functionality of the communication executive due to the relative ease in which additional components can be added to the process. Moreover, support for the use of message validation components facilitates pre-processing of all incoming requests from remote sensors.

[0156] In use, whenever analysis of an unknown material is desired, an operator at a remote sensor performs NIR spectroscopy on the material to generate measurement data, e.g., spectral data. An application running on the associated computer for the NIR instrument then transmits the measurement data over the Internet to web server **234**. Some preprocessing of the measurement data may also be performed prior to transmission to web server **234**.

[0157] In response to the receipt of a request from a remote sensor, the web server instantiates a COM Service component **252**, and invokes a send message function to push the measurement data into the message repository (cube). The COM Service then waits for the status of the message to change. The web server also writes the measurement data to the analysis database at the same time as adding the message to the cube.

[0158] In response to the enqueuing of a new message on the cube, the delegator associated with that message detects a new message and instantiates an appropriate controller component. The controller component subsequently launches one or more processor components as defined in the configuration tables for the cube. The processor components then read the measurement data from the cube, and forward the data to the analysis engine (and optionally, the analysis database as well) to perform chemometric analysis of the data. The request made by the processor is typically in a format that is native to the analysis engine.

[0159] In response to the request, the analysis engine performs the desired analysis of the measurement data, and stores the results in analysis database **246**. In connection with writing the results back to the analysis database, the request in the cube is updated to a "complete" status by the processor, and the web server reads the information from the analysis database based upon a detected "complete" status in the cube. The web server then returns an HTML-compatible web page to the operator, presenting the requested results.

[0160] Another exemplary system incorporating a communication executive consistent with the invention is illustrated at **270** in **FIG. 12**. System **270** is an example of a manufacturing control system in which a communication executive **272** is utilized to interface a manufacturing automation system **274** (incorporating control devices and/or sensor devices), as well as additional sensors such as an NIR sensor **276**, with a manufacturing database **290**. The manufacturing automation system **274** may incorporate a manufacturing process automation system such as the DeltaV automation system available from Fisher-Rosemount.

[0161] Communication executive **272** includes a message repository or cube **276** accessed by one or more instantiations of COM Service **278**. A delegator **280** and controller **282** are also shown. Moreover, a plurality of processors **284**, **286** and **288** are illustrated for performing different tasks in response to messages placed on the cube.

[0162] A manufacturing database **290** is coupled to the processors, as well as to a task assembler **292** and at line processor **294**. An analysis controller, e.g., as described above in connection with **FIG. 11**, may also interface with processor **284** for the purpose of interacting with NIR sensor **276**, which provides additional feedback for a manufacturing process.

[0163] In use, the system of **FIG. 12** operates as a task-based control system, where tasks are performed in response to an indication that a new task is required by either the task assembler **292** or at line processor **294**. Communication with the communication executive is established through the appropriate COM Service **278**, which writes a request to the cube **276** as appropriate. The respective task assembler or at line processor then monitors the progress of the message through the COM Service.

[0164] In response to the placement of a new message on the cube, the delegator that is responsible for that type of message reads the new message from the cube, and instantiates a controller responsible for handling such messages. The controller then updates the status of the message to "in process," and based upon the settings in the database, calls one or more processors **284**, **286** and **288** to handle the message as appropriated, passing data as needed. Processors **284** and **288** have the ability to communicate with manufacturing database **290** to retrieve appropriate information, as well as write the results to the database. For example, processor **288** may log events to the manufacturing database through its operation as an event log controller. Likewise, processor **284** may perform the function of an analysis request handler for interfacing with analysis controller **296**. Processor **286** may function as a spreadsheet handler, e.g., to perform complex calculations.

[0165] Once the processors have completed processing of a current message, the controller associated with that message updates the status of the message to "complete." The calling system (e.g., task assembler **292** or at line processor **294**) will detect the "complete" status through the COM Service, and read any necessary information from the manufacturing database. The task assembler **292** may then interact with the manufacturing automation system based upon the new information.

[0166] Yet another example embodiment within which a communication executive may find use is an inventory management system, as shown at **300** in **FIG. 13**. System **300** is an example of an inventory management system in which a communication executive **302** is utilized to interface

one or more inventory measurement sensors **304** with an inventory database **306**. Each sensor **304** may be implemented using any device capable of detecting a particular type of inventory (e.g., a fluid level, a particulate level, etc.), coupled with functionality for transmitting the sensed data for collection by the inventory management system. For example, sensors **304** may be grouped as illustrated at **308** into a tier of measurement devices located at supplier sites. Other sensors **304** may be coupled to a data consolidator **310** using third party technology. In either event, data generated by the sensors **304** is output over a communications network such as the Internet **312** to an Internet webserver **314**, e.g., providing ASP pages suitable for receiving the data in an appropriate format. For example, sensors **304** and/or consolidator **310** may be configured to generate XML data streams, and to post such XML-formatted data to an ASP page resident on the web server. Consolidator **310**, for example, may be capable of pooling the data from multiple sensors **304** into a consolidated XML document.

[**0167**] In other embodiments, some or all of the sensors **304** may be coupled to webserver **314** via other communication networks. Moreover, in other embodiments, a non-Internet protocol implementation may be utilized, whereby webserver **314** may not be utilized.

[**0168**] Communication executive **302** includes one or more instantiations of COM Service **320** that submit messages to a message repository or cube **322**. A delegator **324** and controller **326** are also shown. Moreover, consistent with the invention, one or more processors **328** are dynamically instantiated to perform actions in connection with inventory database **306** in response to messages placed on the cube.

[**0169**] In use, the system of **FIG. 13** collects inventory data from sensors **304** and forwards the data to database **306**. In addition, processor **328**, or alternatively, other program code, may be configured to track current inventories and, as necessary, notify purchasing personnel and/or automatically order or schedule the delivery of additional inventory. Moreover, in some implementations feedback information may be provided to the various sites served by sensors **304**.

[**0170**] Other modifications will be apparent to one of ordinary skill in the art. Therefore, the invention lies in the claims hereinafter appended.

What is claimed is:

1. An apparatus configured to interface communications between sending and receiving electronic devices, the apparatus comprising:

(a) a message buffer configured to receive messages from at least one sending electronic device, each message associated with at least one message type among a plurality of message types;

(b) a plurality of processor components, each configured to perform a task associated with a message received by the message buffer, wherein at least one processor component is configured to initiate an action on a receiving electronic device;

(c) a plurality of controller components, each associated with a message type among the plurality of message types, and each configured to handle a message associated with such associated message type by dynamically instantiating at least a subset of the plurality of processor components; and

(d) at least one delegator component associated with at least one message type among the plurality of message types and configured to monitor the message buffer for messages associated with such associated message type, the delegator component further configured to dynamically execute a controller component associated with such associated message type in response to detecting an addition of a message associated with such associated message type to the message buffer.

2. The apparatus of claim 1, wherein the receiving electronic device comprises a resource.

3. The apparatus of claim 1, wherein the sending electronic device comprises a near infrared sensor, and wherein the receiving electronic device comprises an analysis engine.

4. The apparatus of claim 3, wherein the apparatus is coupled to the sending electronic device via the Internet.

5. The apparatus of claim 1, wherein the delegator component is further configured to be selectively enabled and disabled in response to user input, and wherein the delegator component is configured to dynamically execute the controller component only if the delegator component is enabled when the message is added to the message buffer.

6. The apparatus of claim 5, further comprising a buffer services component configured to add the message to the message buffer irrespective of whether the delegator component is enabled.

7. The apparatus of claim 1, wherein the delegator component is further configured to associate the message with the delegator component to restrict another delegator component from handling the message.

8. The apparatus of claim 1, wherein the delegator component is further configured to dynamically instantiate the controller component in response to detecting the addition of the message to the message buffer.

9. The apparatus of claim 1, wherein each controller component is further configured to retrieve from the message buffer a list of messages awaiting processing by the controller component.

10. The apparatus of claim 1, wherein each controller component is further configured to determine which of the plurality of processor components need to be executed to handle a message.

11. The apparatus of claim 10, further comprising a database that maintains a mapping of processor components to controller components, wherein each controller component is configured to determine which of the plurality of processor components need to be executed to handle a message by accessing the database.

12. The apparatus of claim 10, wherein a first controller component is configured to sequentially execute first and second processor components when handling a message associated therewith.

13. The apparatus of claim 10, wherein a first controller component is configured to concurrently execute first and second processor components when handling a message associated therewith.

14. The apparatus of claim 1, wherein a first processor component among the plurality of processor components is configured to batch process a plurality of messages stored in the message buffer and associated with a first message type.

15. The apparatus of claim 1, wherein a first processor component among the plurality of processor components is configured to retrieve message data from a message record stored in the message buffer in connection with initiating an action with a receiving electronic device.

16. The apparatus of claim 15, wherein the first processor component among the plurality of processor components is configured to return result data by updating the message record stored in the message buffer.

17. The apparatus of claim 1, further comprising:

(a) a plurality of message validation components, each message validation component associated with at least one of a message type and a receiving electronic device, each message validation component configured to validate a message sent to the message buffer by a sending electronic device prior to addition of the message to the message buffer, and each message validation component further configured to return a result that indicates whether the message should be accepted by the message buffer; and

(b) a buffer services component configured to process a first message sent to the message buffer by invoking a message validation component associated with the first message and selectively adding the first message to the message buffer based upon the result returned by the invoked message validation component.

18. A method of communicating between sending and receiving electronic devices, the method comprising:

(a) actively monitoring a message buffer for addition of a message associated with a first message type among a plurality of message types;

(b) in response to detecting addition of a message associated with the first message type, dynamically executing a first controller component among a plurality of controller components that is associated with the first message type; and

(c) after the first controller component is executed, dynamically instantiating with the first controller component at least a subset of a plurality of processor components to initiate an action on a receiving electronic device.

19. The method of claim 18, wherein the sending electronic device comprises a near infrared sensor, and wherein the receiving electronic device comprises an analysis engine.

20. The method of claim 18, further comprising receiving the message from the sending electronic device via the Internet.

21. The method of claim 18, wherein actively monitoring the message buffer and dynamically executing the first controller component are performed by a delegator component, wherein the delegator component is further configured to be selectively enabled and disabled in response to user input, and wherein dynamically executing the first controller component is performed only if the delegator component is enabled when the message is added to the message buffer.

22. The method of claim 21, further comprising adding the message to the message buffer irrespective of whether the delegator component is enabled.

23. The method of claim 21, further comprising, in response to detecting addition of the message, associating the message with the delegator component to restrict another delegator component from handling the message.

24. The method of claim 18, further comprising dynamically instantiating the first controller component in response to detecting the addition of the message to the message buffer.

25. The method of claim 18, further comprising retrieving, with the first controller component, a list of messages from the message buffer that are awaiting processing by the first controller component.

26. The method of claim 18, further comprising determining with the first controller component which of the plurality of processor components need to be executed to handle the message.

27. The method of claim 26, wherein determining which of the plurality of processor components need to be executed includes accessing a database that maintains a mapping of processor components to controller components.

28. The method of claim 26, wherein dynamically instantiating the subset of processor components includes sequentially executing first and second processor components.

29. The method of claim 26, wherein dynamically instantiating the subset of processor components includes concurrently executing first and second processor components.

30. The method of claim 18, further comprising batch processing a plurality of messages stored in the message buffer and associated with the first message type with a first processor component among the subset of processor components.

31. The method of claim 18, further comprising retrieving message data from a message record stored in the message buffer in connection with initiating an action with a receiving electronic device with a first processor component among the subset of processor components.

32. The method of claim 31, further comprising returning result data with the first processor component by updating the message record stored in the message buffer.

33. The method of claim 18, further comprising:

(a) receiving the message from a sending electronic device;

(b) in response to receiving the message, determining whether any of a plurality of message validation components is associated with at least one of the first message type and the receiving electronic device;

(c) invoking any associated message validation component to validate the message prior to addition of the message to the message buffer, wherein each such associated message validation component returns a result that indicates whether the message should be accepted by the message buffer; and

(d) selectively adding the message to the message buffer based upon the result returned by each associated message validation component.

34. A program product, comprising:

(a) a program configured to interface communications between sending and receiving electronic devices, the program including:

(i) a message buffer configured to receive messages from at least one sending electronic device, each message associated with at least one message type among a plurality of message types;

(ii) a plurality of processor components, each configured to perform a task associated with a message received by the message buffer, wherein at least one processor component is configured to initiate an action on a receiving electronic device;

(iii) a plurality of controller components, each associated with a message type among the plurality of message types, and each configured to handle a message associated with such associated message type by dynamically instantiating at least a subset of the plurality of processor components; and

(iv) at least one delegator component associated with at least one message type among the plurality of message types and configured to monitor the message buffer for messages associated with such associated message type, the delegator component further configured to dynamically execute a controller component associated with such associated message type in response to detecting an addition of a message associated with such associated message type to the message buffer; and

(b) a signal bearing medium bearing the program.

35. An apparatus configured to interface communications between sending and receiving electronic devices, the apparatus comprising:

(a) a message buffer configured to receive messages from at least one sending electronic device, each message associated with at least one message type among a plurality of message types;

(b) a plurality of message validation components, each message validation component associated with at least one of a message type and a receiving electronic device, each message validation component configured to validate a message sent to the message buffer by a sending electronic device prior to addition of the message to the message buffer, and each message validation component further configured to return a result that indicates whether the message should be accepted by the message buffer; and

(c) a buffer services component configured to process a first message sent to the message buffer by invoking a message validation component associated with the first message and selectively adding the first message to the message buffer based upon the result returned by the invoked message validation component.

36. The apparatus of claim 35, wherein the invoked message validation component is configured to selectively modify the first message prior to the first message being added to the message buffer.

37. The apparatus of claim 35, wherein the invoked message validation component is configured to reject the first message by returning a result that indicates that the message should not be accepted by the message buffer, and wherein the buffer services component is configured to prevent the first message from being added to the message buffer in response to the result indicating that the message should not be accepted by the message buffer.

38. A method of communicating between sending and receiving electronic devices, the method comprising:

(a) receiving a message from a sending electronic device, the message directed to a receiving electronic device and associated with a message type among a plurality of message types;

(b) in response to receiving the message, determining whether any of a plurality of message validation components is associated with at least one of the message type and receiving electronic device for the message;

(c) invoking any associated message validation component to validate the message sent to the message buffer by a sending electronic device prior to addition of the message to the message buffer, wherein each such associated message validation component returns a result that indicates whether the message should be accepted by the message buffer; and

(d) selectively adding the message to the message buffer based upon the result returned by each associated message validation component.

39. A program product, comprising:

(a) a program configured to interface communications between sending and receiving electronic devices, the program including:

(i) a message buffer configured to receive messages from at least one sending electronic device, each message associated with at least one message type among a plurality of message types;

(ii) a plurality of message validation components, each message validation component associated with at least one of a message type and a receiving electronic device, each message validation component configured to validate a message sent to the message buffer by a sending electronic device prior to addition of the message to the message buffer, and each message validation component further configured to return a result that indicates whether the message should be accepted by the message buffer; and

(iii) a buffer services component configured to process a first message sent to the message buffer by invoking a message validation component associated with the first message and selectively adding the first message to the message buffer based upon the result returned by the invoked message validation component; and

(b) a signal bearing medium bearing the program.

40. A data processing system configured to process spectral data generated by a plurality of remote sensing stations during sampling of unknown materials, the data processing system comprising:

(a) an analysis engine configured to process spectral data to generate quantitative results therefrom using multivariate analysis; and

(b) a communication executive interposed between the plurality of remote sensing stations and the analysis engine, the communication executive configured to communicate first spectral data generated by a first remote sensing station to the analysis engine, and to communicate first quantitative results from the analysis engine to the first remote sensing station subsequent to

processing of the first spectral data by the analysis engine, the communication executive comprising:

(i) a message buffer configured to receive messages including spectral data generated by the plurality of remote sensing stations;

(ii) a plurality of processor components, each configured to perform a task associated with a message received by the message buffer, at least one processor component configured to access the analysis engine to initiate processing of spectral data by the analysis engine;

(iii) a plurality of controller components, each configured to dynamically instantiate at least a subset of the plurality of processor components; and

(iv) at least one delegator component configured to monitor the message buffer for messages, and in response to a first message associated with the first spectral data, to dynamically execute a controller component so as to initiate analysis of the first spectral data by the analysis engine and return the first quantitative results to the first remote sensing station.

41. The data processing system of claim 40, further comprising a web server configured to interface with the plurality of remote sensing stations over the Internet, the web server configured to add the first message to the message buffer in response to receipt of the first spectral data from the first remote sensing station, and to communicate the first quantitative results to the first remote sensing station when the first quantitative results are returned by the analysis engine.

* * * * *