(12) STANDARD PATENT (11) Application No. **AU 2010213618 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Managing task execution

(51) International Patent Classification(s)
G06F 9/46 (2006.01)

(21) Application No: **2010213618** (22) Date of Filing: **2010.02.12**

(87) WIPO No: **WO10/093879**

(30) Priority Data

(31) Number	(32) Date	(33) Country
61/152,669	**2009.02.13**	**US**

(43) Publication Date: **2010.08.19**
(44) Accepted Journal Date: **2015.07.02**

(71) Applicant(s)
Ab Initio Technology LLC

(72) Inventor(s)
Buxbaum, Mark;Wakeling, Tim

(74) Agent / Attorney
Pizzeys, PO Box 291, WODEN, ACT, 2606

(56) Related Art
US 2004/0073529 A1 (STANFILL) 15 April 2004
VAN DER AALST et al.;"Workflow Patterns"; Distributed and Parallel Databases . Kluwer Academic Publishers, Vol. 14. no. 1. 1 July 2003
US 5357632 A (PIAN et al.) 18 October 1994

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 August 2010 (19.08.2010)

PCT

(10) International Publication Number
WO 2010/093879 A1

(51) International Patent Classification:
G06F 9/46 (2006.01)

(21) International Application Number:
PCT/US2010/024036

(22) International Filing Date:
12 February 2010 (12.02.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/152,669 13 February 2009 (13.02.2009) US

(71) Applicant *(for all designated States except US)*: **AB INITIO TECHNOLOGY LLC** [US/US]; 201 Spring Street, Lexington, Massachusetts 02421 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BUXBAUM, Mark** [US/US]; 32 Brucewood Road, Acton, Massachusetts 01720 (US). **WAKELING, Tim** [GB/US]; 11 Abbot Street, Andover, Massachusetts 01810 (US).

(74) Agents: **HENNESSEY, Gilbert H.** et al.; Fish & Richardson P.C., P.O. Box 1022, Minneapolis, Minnesota 55440-1022 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

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

— *with amended claims (Art. 19(1))*

(54) **Title**: MANAGING TASK EXECUTION

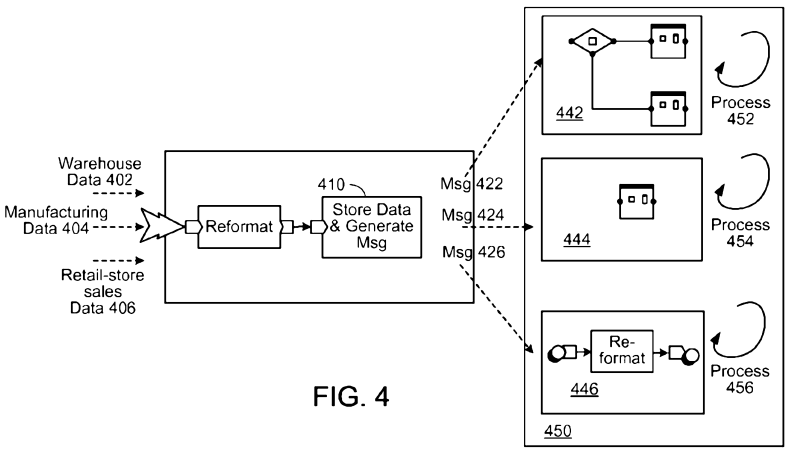

FIG. 4

(57) **Abstract**: Managing task execution includes: receiving a specification of a plurality of tasks (442, 444, 446) to be performed by respective functional modules; processing a flow of input data (402, 404, 406) using a dataflow graph that includes nodes representing data processing components (410) connected by links representing flows of data between data processing components; in response to at least one flow of data provided by at least one data processing component (410), generating a flow of messages (422); and in response to each of the messages in the flow of messages, performing an iteration of a set of one or more tasks (442) using one or more corresponding functional modules.

WO 2010/093879 A1

MANAGING TASK EXECUTION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Application Serial No. 61/152,669, filed on February 13, 2009, incorporated herein by reference.

BACKGROUND

This description relates to managing task execution.

A data processing system typically stores, manages, processes, and analyzes data. A data processing system may simply handle one or more tasks scheduled to be completed at a certain time. Alternatively, a data processing system may be a large-scale customer relationship management system. Simple or complex, a data processing system often includes one or more components, such as processing elements, input data, and output data. The processing elements of a data processing system determine the function of the data processing system, for example, data warehouse, customer relationship management, and data mining, etc.. Input data of a data processing system can come from many sources. For example, input data may come from flat files, database tables, operational systems, etc.. Input data may also come from the Internet, carrying information from one system to another. Output data of a data processing system are what the processing elements generate. Formats of output data vary depending on the processing elements that generate them.

SUMMARY

In one aspect, in general, a method for managing task execution includes receiving a specification of a plurality of tasks to be performed by respective functional modules; processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components; in response to at least one flow of data provided by at least one data processing component, generating a flow of messages; and in response to each of the messages in the flow of messages, performing an iteration of a set of one or more tasks using one or more corresponding functional modules.

Aspects can include one or more of the following features.

At least one of the functional modules is configured to initiate execution of the dataflow graph.

The specification of the plurality of tasks specifies dependency relationships between the at least two of the tasks.

A dependency relationship between the at least two tasks defines at least a partial ordering for execution of the functional modules corresponding to the tasks.

A dependency relationship between the at least two tasks defines conditional logic for determine at least one condition upon which execution of at least one of the functional modules is based.

At least one of the functional modules includes a fault-handling module that is executed when the conditional logic detects that a fault has occurred in execution of one of the other functional modules.

Multiple iterations of a given set of one or more tasks are executed concurrently in response to two or more messages in the flow of messages.

One or more of the messages in the flow of messages is generated in response to an element of data in the flow of data without including the element of data.

One or more of the messages in the flow of messages includes at least a portion of an element of data in the flow of data.

At least one of the functional modules is configured to send an acknowledgement in response to receiving one of the messages in the flow of messages.

At least one of the data processing components resends an unacknowledged message.

The method further includes storing a parameter value identifying the specification of a plurality of tasks.

The method further includes transmitting the generated flow of messages to an application for receiving the messages identified by the parameter value.

The method further includes storing messages received by the application in parameters visible to multiple processes for performing the tasks.

In another aspect, in general, a system for managing task execution includes: a task managing system including circuitry for receiving a specification of a plurality of tasks to be performed by respective functional modules; and a data processing system including circuitry for processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components. The data processing system is configured to generate a flow of messages in response to at least one flow of data

provided by at least one data processing component. The task managing system is configured to perform an iteration of a set of one or more tasks using one or more corresponding functional modules in response to each of the messages in the flow of messages.

In another aspect, in general, a system for managing task execution includes: means for receiving a specification of a plurality of tasks to be performed by respective functional modules; and means for processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components. The data processing system is configured to generate a flow of messages in response to at least one flow of data provided by at least one data processing component. The task managing system is configured to perform an iteration of a set of one or more tasks using one or more corresponding functional modules in response to each of the messages in the flow of messages.

In another aspect, in general, a computer-readable medium stores a computer program for managing task execution. The computer program includes instructions for causing a computer to: receive a specification of a plurality of tasks to be performed by respective functional modules; process a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components; in response to at least one flow of data provided by at least one data processing component, generate a flow of messages; and in response to each of the messages in the flow of messages, perform an iteration of a set of one or more tasks using one or more corresponding functional modules.

Aspects can include one or more of the following advantages.

The techniques enable data flow to be converted into control flow and can facilitate data processing situations where incoming data are continuous and unpredictable and each piece of data may need elaborate handling.

Dataflow graphs can be incorporated into the control flow of a task managing application, allowing different dataflow graphs for data processing based on the values stored in the incoming message generated in response to elements of a data flow.

Having separate development environments for data processing and task management allows development of data processing applications and task managing applications to be sandboxed into independent environments that do not interfere with each other.

Since data processing applications often emphasize data availability, data transformation, and data integrity, and task managing applications often emphasize error handling, system resource allocation, and computation order, using separate graphical development tools in a complex data processing system for developing data processing applications and task managing applications allow each tool to meet the unique requirements of each type of the applications.

Having a separate data processing application and task managing application also facilitates software reuse.

In a complex data processing system, data may come from diverse external sources and take on different formats. Incoming data may be corrupted and error checking may be used to ensure data integrity. A separate data processing application that handles reformatting and error checking encapsulates and isolates this complexity from a

downstream task managing application, allowing task managing application to be developed without specific knowledge of possible data sources and to be reused when data sources or formats are changed. Likewise data processing applications can be developed with a focus on the data sources and without specific knowledge of the downstream computation environment and can be reused even when downstream handling has been changed.

Other features and advantages of the invention will become apparent from the following description, and from the claims.

DESCRIPTION OF DRAWINGS

FIGs.1A and 1B show examples of data processing systems.

FIG. 2 is an example of a dataflow graph.

FIG. 3 is an example of a control flow diagram.

FIG. 4 illustrates how to convert data flow to control flow.

FIG. 5 is a block diagram of a data processing system.

FIG. 6 is an example showing how to configure an application to run iteratively.

FIGs. 7A and 7B are block diagrams showing implementations of iterative task handling.

FIG. 8 is an example of a data transmitting mechanism.

DETAILED DESCRIPTIONS

FIGs. 1A and 1B show examples of data processing systems. The data processing system 100 in FIG. 1A is a chart generation tool. In FIG. 1A, a Java program 104 reads input from a table 102 that contains sales data, then plots a bar chart 106 as output. The

Java program 104 is a processing element of the data processing system 100; the table 102 is the input data; and the bar chart 106 is the output data.

The data processing system 120 in FIG. 1B includes a call center 130 deployed in a retail business. The call center 130 pulls data from a warehouse 122, a manufacturer 124, a retail store 126, and terminals 134 to give call center agents 131, 132, 133, and 134 real-time information regarding merchandise availability. Servers 136, the terminals 134, and the applications running on them are the processing elements. Information pulled from the warehouse 122, the manufacturer 124, the retail store 126, and the terminals 134 is input data. Real-time information regarding merchandise availability that is displayed on the call agents' terminals is output data.

A complex data processing system, such as the one shown in FIG. 1B, handles volumes of data and performs myriads of computations through applications that run on the system. Some applications are commercially available. Some applications have to be customer tailored to meet customers' special demands.

Applications in a complex data processing system tend to specialize. For example, some applications may mainly handle data processing and some may mainly handle task managing. Data processing applications usually handle data-related computations, such as reformatting, sorting, and organizing data. Data processing applications tend to have a focus on how data flow from a source to a destination and how data are transformed in the process. Task managing applications usually handle scheduling and initiating execution of computation-related jobs, such as executing programs, scheduling events, managing processes, and dealing with faulty conditions. Task managing applications

tend to have an emphasis on how control flows from task to task and how control flow is affected by conditional logic and faulty conditions.

Data processing applications and task managing applications typically have different characteristics. Development environments for these two types of applications may be the same or separate. An exemplary development environment for a data processing application is a graph-based computation environment described below.

A graph-based computation environment allows a programmer to build a graph-based application by using components as building blocks. A graph-based application is often represented by a directed graph, with nodes (or "vertices") in the graph representing components (either data storage components or executable computation components), and the directed links (or "edges") in the graph representing flows of data between components. A dataflow graph (also called simply a "graph") is a modular entity. Each graph can be made up of one or more other graphs, and a particular graph can be a component in a larger graph. A graphical development environment (GDE) provides a user interface for developers to develop, test, and deploy applications as executable graphs that are tailored to a given user-specific environment.

FIG. 2 is an example of a dataflow graph that represents a data processing application. In FIG. 2, input dataset component 202 contains status data for a particular merchandise. The status data are pulled from warehouses, manufacturers, and retail stores. A computation component 204 reads the data contained in the input dataset component 202, reformats them, and feeds them to a next computation component 208. The computation component 208 analyzes the reformatted data and generates output data to be stored in output dataset component 210. The output data represent merchandise

availability. Symbol 212 is an output port representing a source of data elements (e.g., records or other units of data) flowing out of the input dataset component 202. Symbol 214 is an input port representing a destination for data elements flowing into the computation component 208. Link 206 shows the direction in which the data flows, i.e., from the computation component 204 to the computation component 208. The data processing application embodied by the dataflow graph 200 can be used in a call center system, such as the call center system 120 shown in FIG. 1B.

A dataflow graph, such as graph 200, embodies the data processing aspects of a data processing application such as, for example, where the data come from, what transformations the data undergo, and where the data arrive.

As a comparison, FIG. 3 shows a control flow diagram 300 (also called a "Plan"). The control flow diagram 300 represents a specification for a task managing application which specifies tasks to be performed by respective functional modules. A functional module may perform a task that includes, for example, a series of actions to be executed on a computer system. A functional module may be implemented using a dataflow graph or may represent another control flow diagram (called a "SubPlan"). A functional module may be an implementation of conditional logic, or a program, or a user script. In the control flow diagram 300, functional modules, such as Demographics Graph 302, Mailing Graph 306, Top Customers Graph 310, and Bottom Customers Graph 314, are dataflow graphs.

Load Transactions Program 304 is a functional module that includes executable code e.g., a script in Python or Perl. Distribute Mail Conditional Task 308 is an implementation of conditional logic, directing the control to either a Top Customers

Graph 310, or a Credit SubPlan 312, or a Bottom Customers Graph 314 depending on whether a particular customer is classified as a top customer or a bottom customer or a customer who needs credit extension. Credit SubPlan 312 and Charge SubPlan 316 are themselves control flow diagrams. The functional modules are connected with arrow links, 322, 324, etc.. The arrow links specify dependency relationships among the tasks performed by the functional modules and thus indicate how control flows from one functional module to another and define at least a partial ordering according to which the functional modules run. For example, Demographic Graph 302 is run before Mailing Graph 306.

A control flow diagram, such as diagram 300, embodies the control aspects of the task managing application controlled by the control flow diagram. Control aspects of a task managing application include, for example, determining running sequences among different tasks and applying conditional logic.

As mentioned above, a complex data processing system may need to manage large volumes of data as well as perform numerous computations. In a complex data processing system, both task managing applications and data processing applications may be used.

For example, in the call center system 120 shown in FIG. 1B, customer transaction data stream into the system and are processed using real-time processing. If only a one-step processing is required to handle the data, such as registering the data as an entry in a database, a simple application represented by a dataflow graph such as the dataflow graph 200 may be sufficient. However, in many cases, multi-step processing may be needed. For example, in a large retail business, thousands of customer

transactions may happen simultaneously. Parallel processing may be needed. Or in a sophisticated retail business, customers may be accorded differential treatments. Therefore conditional logic may be implemented to handle different classes of customers. Or in a reliable call center system, exception handling, failover, and data clean-up may be used. Thus, fault handling may be required.

When data processing is a multi-step process that involves parallel processing, conditional logic, and/or fault handling, a simple dataflow graph may not be the best approach to capture the complexity of the multi-step process. It may be advantageous to convert data flow to control flow at a certain stage during the process.

For example, a complex data processing system, such as the call center system 120 shown in FIG. 1B, may use both data processing applications and task managing applications. Incoming data are first processed in a data processing application. The data is then used to drive the control flow in a task managing application. For a developer who develops applications for a complex data processing system, the technique of letting the data flow drive the control flow can be used, for example, when incoming data are streaming, continuous, and unpredictable, or when concurrent data processing may be needed. An example of this technique is illustrated in FIG. 4.

In FIG. 4, a data processing application (such as the call center system 120 shown in FIG.1B) receives a flow of input data, such as warehouse data 402, manufacturing data 404, and retail-store sales data 406, from various external sources. As the input data stream into the data processing application, implemented in this example by the dataflow graph 410, a component in the graph 410 reformats the data coming from different sources into an appropriate format. Another component in the graph 410 then stores the

data and generates messages to be sent to the task managing application 450. Messages generated by the data processing application are based on the received incoming data or an element in the received incoming data, and in some implementations portions of the incoming data can be used as the messages.

Different mechanisms can be used to transmit a message from the data processing application 410 to the task managing application 450, for example, message passing, queues, shared memory space, remote procedure calls. The task managing application 450, based on the messages or certain values contained in the messages, can invoke different processes, such as a process executing a Plan 442, a SubPlan 444, and/or a dataflow graph 446.

In some cases the task managing application 450 invokes a separate iteration of a corresponding set of one or more tasks (e.g., Plan 442, SubPlan 444, or dataflow graph 446) for each incoming message received from the data processing application 410. So if each message is generated in response to an element of data generated as output of the data processing application 410, the application 450 is able to iterate for each element of data. A loop index increments for each iteration of the task managing application 450. In each loop iteration, a process associated with the current loop index is spun off to handle an incoming message. Depending on the message received by the task managing application 450 or a value contained in the message, the process spun off to handle an incoming message can execute a dataflow graph, or a SubPlan, or any program for performing a set of one or more tasks.

In the illustrated example, for the first element of incoming warehouse data 402 processed by the data processing application 410, a message 422 is generated by the data

processing application 410 and transmitted to the task managing application 450. The task managing application 450 being in its first loop iteration (with loop index 0), spins off a child process 452 to handle the message 442. The process 452 corresponds to the Plan 442, evoked by the task managing application 450 to reduce the number of merchandise available. A child process 454 is initiated in a second loop iteration associated with loop index 1 of the task managing application 450 to handle the second element of incoming data, generated in response to the manufacturing data 404. The child process 454 may correspond to the SubPlan 444, which, for example, performs the task of increasing the number of merchandise available. A child process 456 is initiated in a third loop iteration associated with loop index 2 to handle the third element of incoming data, generated in response to the retail store sales data 406. The child process 456 may correspond to executing the dataflow graph 446. The task managing application may be configured to invoke the processes, 452, 454, and 456, concurrently or serially.

Processes may refer to processes that run on different CPUs or on the same CPU. In the latter case, the processes may also be known as "threads".

For increased reliability, the task managing application 450 may be configured to send an acknowledgement to the data processing application 410 when it receives a message. The acknowledgement can be a positive acknowledgement if the task managing application decides that the message received is intact or a negative one if the task managing application decides that the message received is corrupted.

The data processing application 410 can be configured to wait for an acknowledgement that the last-sent message has been received before sending the next message. It can be further configured to send the next message upon receiving a positive

acknowledgement and re-send the last message upon receiving a negative acknowledgement.

FIG. 5 is a block diagram of an application development and execution system 500. The system 500 includes two graphical development environments, a GDE 512 used for dataflow graph development and a GDE 514 used for control flow diagram development. Alternatively, one graphical development environment can be used for development of both dataflow graphs and control flow diagrams. Or a graphical development environment can be used for developing control flow diagrams and a command line user interface can be used for dataflow graphs, or vice versa.

Using the GDE 512, a data processing application 518 is built that includes a dataflow graph 580. Using the GDE 514, a task managing application 516 is built that includes a control flow diagram 550.

The system 500 also includes a database 520. The database 520 may be a scalable object-oriented database system that provides storage for various kinds of information (e.g., metadata) for the system 500. The database 520 may be, for example, an enterprise metadata database, which can support the development and execution of graph-based applications and the interchange of data between the graph-based applications and other systems, e.g., operating systems.

The system 500 further includes an operating system 524. The operating system 524 may be, for example, a parallel operating environment. The operating system 524 provides support for running application development environments, such as GDE 512 and GDE 514, and provides for scalable parallel and distributed execution of the applications developed.

In FIG. 5, a stream of input data 530 go through the dataflow graph 580 as the data are being processed by the data processing application 518. The dataflow graph 580 includes a computation component, reformat 584. In the dataflow graph 580, the data flow from input dataset 582 to output dataset 586.

After being reformatted, the data flow out of the data processing application 518 and into the task managing application 516, and are used to drive the task managing application 516 driven by the control flow diagram 550. The control flow diagram 550 shows two tasks, task 552 and task 554. A task may be a computation performed, for example, by executing a dataflow graph or a script, such as a Perl script. A time sequence 556 shows the running sequence of the tasks specified in the control flow diagram 550. In this case, the task 552 is executed before the task 554.

As shown in FIG. 5, the task 552 in the control flow diagram 550 is a Perl script and the task 554 in the control flow diagram is itself a control flow diagram of sub-tasks. The task 554 includes several sub-tasks implemented by methods that can be executed in the operating system 524. These methods may be existing methods provided by the system or customized methods developed by users. As shown in FIG. 5 as an example, the task 554 includes five methods: Trigger, At Start, Perform, At Failure, and At Success. These methods may be provided by the system or may be developed by a customer.

In some examples, the above five methods may be implemented as follows.

Method Trigger may be implemented to represent the starting point of the task 554. It may contain the condition for starting the execution. The condition may be whether a specific file exists, or whether a flag has been set to true.

Method At Start may be implemented as a method that prepares the system for the method Perform, such as setting environmental variables to desired values, or setting up log files to log runtime information.

Method Perform may be implemented to perform the main functionality of the task 554. Task 554 may also contain conditional logic to handle what happens after method Perform. If method Perform succeeds during its execution, method At Success is executed to exit task 554 with a return code of zero. If method Perform fails during its execution, method At Failure is executed to exit task 554 with a return code of non-zero. Optionally, additional methods can be added for rollback, error handling, and recovery. For example, a method of rollback can be added to roll back what has been done in reverse execution order starting at the point of failure. Alternatively. a method of cleanup can be added to clean up the failed conditions, by resetting flags, registers, etc..

To handle iterative incoming data, a looping SubPlan can be used. In some implementations, a task managing application is configured to include a looping SubPlan. As shown in FIG. 6, the control flow diagram 602 is an example of a looping SubPlan and has conditional logic implemented for handling an element of incoming data. To handle a stream of incoming data, the control flow diagram 602 is run iteratively. The screen shot 604 is an example showing how to configure a simple application, called "My_Subplan" in this example, as a looping SubPlan by setting a predetermined property to "looping" and by setting appropriate looping properties. Looping properties include loop type (Do-While, For-Each, etc.), loop value vector, loop concurrent, loop count and loop indices, etc..

FIGs. 7a and 7b demonstrate, as an example, how to construct a system that includes a task managing application and a data processing application to handle iterative incoming data.

Suppose that we have a business that involves processing customer transactions that arrive continuously and unpredictably. A developer can construct a data processing application 702 to handle data formatting and other preparation work, and a task managing application 704 to perform tasks to further process the data. After the data processing application and the task managing application have been constructed, the data processing application can be configured to pass data to the task managing application and the task managing application can be configured to listen for messages that are coming from the data processing application. In some implementations, messages passed between the data processing application and the task managing application may include data output by the data processing application (e.g., encapsulated and/or encrypted in messages). In some implementations, messages passed between the data processing application and the task managing application can be generated in response to data from the data processing application without including the output data itself. Thus, the term "message" can refer to information passed between the data processing application and the task managing application in any form or format.

On the task managing side, the task managing application 704 includes a looping set of one or more tasks (e.g., a SubPlan) that listens continuously for messages from the data processing application 702. Symbol 706 is a symbol indicating that the application is running iteratively. A message arriving for the task managing application 704 triggers a new loop iteration in which a process can be spun off. The task managing application 704

can be configured to wait until the last loop iteration finishes before starting a new loop iteration or to start a new iteration immediately upon the receipt of a message. In the latter case, processes spun out of each iteration run concurrently.

On the data processing side, a user can configure a message-transmitting application (e.g., a dataflow graph) to "talk" to a counterpart listening application, which in this case is the task managing application 704. In some implementations, the message-transmitting application defines a parameter that holds the name of the counterpart listening application so the message-transmitting application knows where to send messages.

As mentioned before, having a separate data processing application and task managing application provides the advantage of software re-use. However when the task managing application, i.e., the counterpart listening application, has been replaced by a new task managing application, the parameter in the message transmitting application that holds the name of the counterpart listening application needs to be updated correspondingly. A user may need to open the message-transmitting application and make the required change.

To avoid the need of opening the message transmitting application every time the task managing application has been replaced by a new application, parameter Name_of_Listening_Application can be made visible to both the message transmitting application and any counterpart listening application. In the listening application, parameter Name_of_Listening_Application is assigned the value of the listening application's name. Because the parameter is also visible to the message transmitting application, the message transmitting application can read the value of parameter

Name_of_Listening_Application to find out the application to which it is supposed to send message. In this way, the listening application can be changed even at run time without any need of opening the message transmitting application for updates.

In some implementations, the listening application stores the received messages in parameters. A parameter of a process defines a system setting for that process. In some cases, a parameter of a parent process can be inherited by and therefore visible to its child processes. For example, parameter 720 in FIG. 7a are used to store messages transmitted from data processing application 702 to task managing application 704.

Optionally, the user can construct a program or a method on the task managing side to kick off the message-transmitting application, as shown in FIG. 7b. In FIG. 7b, the task managing application 722 (a Plan) includes a listening program 704, which is a task managing application itself (a SubPlan), that listens for messages, and a starter program 726 that initiates execution of the message-transmitting data processing application 702, as shown by an arrow 730.

FIG. 7b also shows that the data processing application 702 includes two data processing applications, 732 and 734, and a queue 736. The data processing application 732 first performs complex processing on incoming data. Then it hands the data over so that the data get published to the queue 736. From the queue 736, the data processing application 734 retrieves the data, reformats them, and publishes them to the task managing application 704 by sending a message to be stored in a predetermined location. Here the queue 736 is acting as a buffer between the incoming data streaming into the data processing application 732 and the outgoing data streaming out of the data processing application 734. In this way, the data processing application 734 may be

configured to wait for an acknowledgement of the previously sent message before sending out the next message without blocking the data processing application 732 from processing incoming data that are streaming in continuously and unpredictably. Also the data processing application 734 is kept simple and lightweight because part of the complex processing is removed from it.

When the task managing application 722 starts running, the starter program 726 initiates the data processing applications 732 and 734. In the meantime, the listening program 704 begins listening for messages from the message-transmitting data processing application 702. In some implementations, the data processing applications 732 and 734 and the task managing application 722 may be configured to run on the same host. The data processing applications 732 and 734 and the task managing application 722 may also include various error-handling methods, such as rollback, recovery, clean-up, and acknowledgements and message tracking that are used to make message transmitting resilient to failures, as demonstrated in FIG. 8.

In FIG. 8, the message-transmitting application 802, labeled as publisher, may be the data processing application 702 as shown in FIG. 7a or the data processing application 734 as shown in FIG. 7b. The receiving application 804, labeled as subscriber, may be the task managing application 704 as shown in FIG. 7a or 7b.

In FIG. 8, messages sent from the publisher are assigned a sequence number. Sequence numbers help the publisher to keep track of the messages sent and facilitate acknowledgements of receipt of the messages by the subscriber. For each message received, the subscriber sends an acknowledgement to the publisher. In the

acknowledgement, the subscriber indicates the sequence number of the received message and whether the message is intact (positive) or corrupted (negative).

When the publisher sends a message with a sequence number X, it may wait for an acknowledgement for the message. When it receives the acknowledgement that contains the sequence number X, it sends the next message of a sequence number X+1 if the acknowledgement is positive or resends the message of the sequence number X if the acknowledgement is negative.

Alternatively, the publisher may send messages without waiting for acknowledgements of previously sent messages. The publisher may store unacknowledged messages and resend the messages if no acknowledgement has been received within a certain period of time. The subscriber can be programmed to ignore messages with the same sequence number so that receiving a repeated message will not cause a problem.

If the system were to crash at some point, the publisher can resend the unacknowledged messages on recovery. The unacknowledged messages can be stored in a persistent storage, such as a disk, if it is desired that the data survive system failures.

Other methods or techniques can be used to ensure that each message is transmitted successfully.

The approach described above can be implemented using software for execution on a computer. For instance, the software forms procedures in one or more computer programs that execute on one or more programmed or programmable computer systems (which may be of various architectures such as distributed, client/server, or grid) each including at least one processor, at least one data storage system (including volatile and

non-volatile memory and/or storage elements), at least one input device or port, and at least one output device or port. The software may form one or more modules of a larger program, for example, that provides other services related to the design and configuration of computation graphs. The nodes and elements of the graph can be implemented as data structures stored in a computer readable medium or other organized data conforming to a data model stored in a data repository.

The software may be provided on a storage medium, such as a CD-ROM, readable by a general or special purpose programmable computer or delivered (encoded in a propagated signal) over a communication medium of a network to the computer where it is executed. All of the functions may be performed on a special purpose computer, or using special-purpose hardware, such as coprocessors. The software may be implemented in a distributed manner in which different parts of the computation specified by the software are performed by different computers. Each such computer program is preferably stored on or downloaded to a storage media or device (e.g., solid state memory or media, or magnetic or optical media) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage media or device is read by the computer system to perform the procedures described herein. The inventive system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer system to operate in a specific and predefined manner to perform the functions described herein.

A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the

spirit and scope of the invention. For example, some of the steps described above may be order independent, and thus can be performed in an order different from that described.

Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

The reference to any prior art in this specification is not, and should not be taken as, an acknowledgement or any form of suggestion that the prior art forms part of the common general knowledge in Australia.

It is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the appended claims. For example, a number of the function steps described above may be performed in a different order without substantially affecting overall processing. Other embodiments are within the scope of the following claims.

2010213618 11 Mar 2013

CLAIMS

1. A method for managing task execution, the method including:

receiving a specification of a plurality of tasks to be performed by respective functional modules;

processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components;

in response to at least one flow of data provided by at least one data processing component, generating a flow of messages; and

in response to each of the messages in the flow of messages, performing an iteration of a set of tasks using corresponding functional modules, with dependency relationships between at least two of the tasks specified by the received specification.

2. The method of claim 1, wherein at least one of the functional modules is configured to initiate execution of the dataflow graph.

3. The method of claim 1, wherein a dependency relationship between the at least two tasks defines at least a partial ordering for execution of the functional modules corresponding to the tasks.

4. The method of claim 1, wherein a dependency relationship between the at least two tasks defines conditional logic for determine at least one condition upon which execution of at least one of the functional modules is based.

5. The method of claim 4, wherein at least one of the functional modules includes a fault-handling module that is executed when the conditional logic detects that a fault has occurred in execution of one of the other functional modules.

6. The method of claim 1, wherein multiple iterations of a given set of one or more tasks are executed concurrently in response to two or more messages in the flow of messages.

7. The method of claim 1, wherein one or more of the messages in the flow of messages is generated in response to an element of data in the flow of data without including the element of data.

8. The method of claim 1, wherein one or more of the messages in the flow of messages includes at least a portion of an element of data in the flow of data.

9. The method of claim 1, wherein at least one of the functional modules is configured to send an acknowledgement in response to receiving one of the messages in the flow of messages.

10. The method of claim9, wherein at least one of the data processing components resends an unacknowledged message.

11. The method of claim 1, further including storing a parameter value identifying the specification of a plurality of tasks.

12. The method of claim 11, further including transmitting the generated flow of messages to an application for receiving the messages identified by the parameter value.

13. The method of claim 12, further including storing messages received by the application in parameters visible to multiple processes for performing the tasks.

14. The method of claim 1, wherein the specification includes a control flow diagram that includes nodes representing the functional modules connected by directed links representing flow of control between functional modules.

15. The method of claim 14, wherein control flows from a first functional module to a second functional module, with a directed link connected from the first functional module to the second functional module, after a task performed by the first functional module has been completed.

16. A system for managing task execution, the system including:

a task managing system including circuitry for receiving a specification of a plurality of tasks to be performed by respective functional modules; and

a data processing system including circuitry for processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components;

wherein the data processing system is configured to generate a flow of messages in response to at least one flow of data provided by at least one data processing component; and

wherein the task managing system is configured to perform an iteration of a set of tasks using corresponding functional modules in response to each of the messages in the flow of messages, with dependency relationships between at least two of the tasks specified by the received specification.

17. The system of claim 16, wherein at least one of the functional modules is configured to initiate execution of the dataflow graph.

18. The system of claim 16, wherein a dependency relationship between the at least two tasks defines at least a partial ordering for execution of the functional modules corresponding to the tasks.

19. The system of claim 16, wherein a dependency relationship between the at least two tasks defines conditional logic for determine at least one condition upon which execution of at least one of the functional modules is based.

20. The system of claim 19, wherein at least one of the functional modules includes a fault-handling module that is executed when the conditional logic detects that a fault has occurred in execution of one of the other functional modules.

21. The system of claim 16, wherein multiple iterations of a given set of one or more tasks are executed concurrently in response to two or more messages in the flow of messages.

22. The system of claim 16, wherein one or more of the messages in the flow of messages is generated in response to an element of data in the flow of data without including the element of data.

23. The system of claim 16, wherein one or more of the messages in the flow of messages includes at least a portion of an element of data in the flow of data.

24. The system of claim 16, wherein at least one of the functional modules is configured to send an acknowledgement in response to receiving one of the messages in the flow of messages.

25. The system of claim9, wherein at least one of the data processing components resends an unacknowledged message.

26. The system of claim 16, wherein the data processing system is configured to store a parameter value identifying the specification of a plurality of tasks.

27. The system of claim 26, wherein the data processing system is configured to transmit the generated flow of messages to an application for receiving the messages identified by the parameter value.

28. The system of claim 27, wherein the data processing system is configured to store messages received by the application in parameters visible to multiple processes for performing the tasks.

29. The system of claim 16, wherein the specification includes a control flow diagram that includes nodes representing the functional modules connected by directed links representing flow of control between functional modules.

30. The system of claim 29, wherein control flows from a first functional module to a second functional module, with a directed link connected from the first

functional module to the second functional module, after a task performed by the first functional module has been completed.

31. A system for managing task execution, the system including:

means for receiving a specification of a plurality of tasks to be performed by respective functional modules;

means for processing a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components;

wherein the data processing system is configured to generate a flow of messages in response to at least one flow of data provided by at least one data processing component; and

wherein the task managing system is configured to perform an iteration of a set of tasks using corresponding functional modules in response to each of the messages in the flow of messages, with dependency relationships between at least two of the tasks specified by the received specification.

32. The system of claim 31, wherein at least one of the functional modules is configured to initiate execution of the dataflow graph.

33. The system of claim 31, wherein a dependency relationship between the at least two tasks defines at least a partial ordering for execution of the functional modules corresponding to the tasks.

34. The system of claim 31, wherein a dependency relationship between the at least two tasks defines conditional logic for determine at least one condition upon which execution of at least one of the functional modules is based.

35. The system of claim 44, wherein at least one of the functional modules includes a fault-handling module that is executed when the conditional logic detects that a fault has occurred in execution of one of the other functional modules.

36. The system of claim 31, wherein multiple iterations of a given set of one or

more tasks are executed concurrently in response to two or more messages in the flow of messages.

37. The system of claim 31, wherein one or more of the messages in the flow of messages is generated in response to an element of data in the flow of data without including the element of data.

38. The system of claim 31, wherein one or more of the messages in the flow of messages includes at least a portion of an element of data in the flow of data.

39. The system of claim 31, wherein at least one of the functional modules is configured to send an acknowledgement in response to receiving one of the messages in the flow of messages.

40. The system of claim 39, wherein at least one of the data processing components resends an unacknowledged message.

41. The system of claim 31, further including means for storing a parameter value identifying the specification of a plurality of tasks.

42. The system of claim 41, further including means for transmitting the generated flow of messages to an application for receiving the messages identified by the parameter value.

43. The system of claim 42, further including means for storing messages received by the application in parameters visible to multiple processes for performing the tasks.

44. The system of claim 31, wherein the specification includes a control flow diagram that includes nodes representing the functional modules connected by directed links representing flow of control between functional modules.

45. The system of claim 44, wherein control flows from a first functional

module to a second functional module, with a directed link connected from the first functional module to the second functional module, after a task performed by the first functional module has been completed.

46. A computer-readable medium storing a computer program for managing task execution, the computer program including instructions for causing a computer to:

receive a specification of a plurality of tasks to be performed by respective functional modules;

process a flow of input data using a dataflow graph that includes nodes representing data processing components connected by links representing flows of data between data processing components;

in response to at least one flow of data provided by at least one data processing component, generate a flow of messages; and

in response to each of the messages in the flow of messages, perform an iteration of a set of tasks using corresponding functional modules, with dependency relationships between at least two of the tasks specified by the received specification.

47. The medium of claim 46, wherein at least one of the functional modules is configured to initiate execution of the dataflow graph.

48. The medium of claim 46, wherein a dependency relationship between the at least two tasks defines at least a partial ordering for execution of the functional modules corresponding to the tasks.

49. The medium of claim 46, wherein a dependency relationship between the at least two tasks defines conditional logic for determine at least one condition upon which execution of at least one of the functional modules is based.

50. The medium of claim 49, wherein at least one of the functional modules includes a fault-handling module that is executed when the conditional logic detects that a fault has occurred in execution of one of the other functional modules.

51. The medium of claim 46, wherein multiple iterations of a given set of one or more tasks are executed concurrently in response to two or more messages in the flow of messages.

52. The medium of claim 46, wherein one or more of the messages in the flow of messages is generated in response to an element of data in the flow of data without including the element of data.

53. The medium of claim 46, wherein one or more of the messages in the flow of messages includes at least a portion of an element of data in the flow of data.

54. The medium of claim 46, wherein at least one of the functional modules is configured to send an acknowledgement in response to receiving one of the messages in the flow of messages.

55. The medium of claim 55, wherein at least one of the data processing components resends an unacknowledged message.

56. The medium of claim 46, wherein the computer program further includes instructions for causing a computer to store a parameter value identifying the specification of a plurality of tasks.

57. The medium of claim 56, wherein the computer program further includes instructions for causing a computer to transmit the generated flow of messages to an application for receiving the messages identified by the parameter value.

58. The medium of claim 57, wherein the computer program further includes instructions for causing a computer to store messages received by the application in parameters visible to multiple processes for performing the tasks.

59. The medium of claim 46, wherein the specification includes a control flow diagram that includes nodes representing the functional modules connected by directed links representing flow of control between functional modules.

60. The medium of claim 59, wherein control flows from a first functional module to a second functional module, with a directed link connected from the first functional module to the second functional module, after a task performed by the first functional module has been completed.

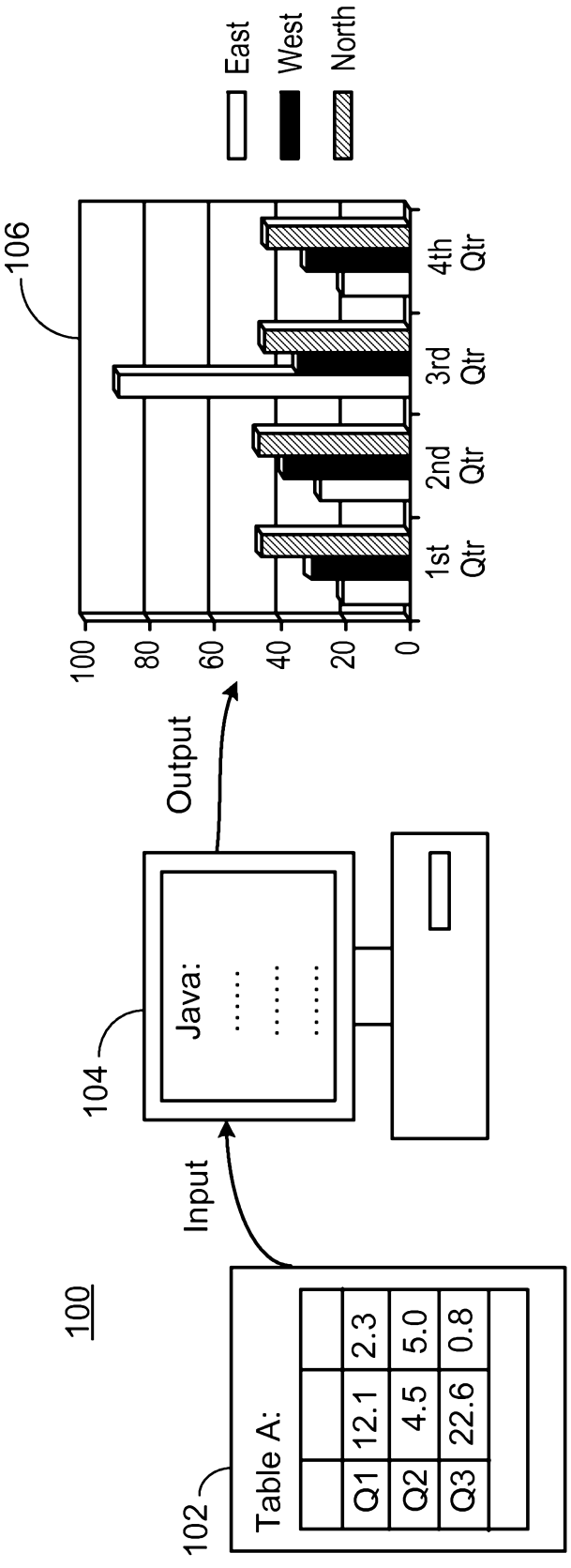
100
102
Table A:
Q1 12.1 2.3
Q2 4.5 5.0
Q3 22.6 0.8
Input
104
Java:
Output
106
East
West
North
100
80
60
40
20
0
1st Qtr
2nd Qtr
3rd Qtr
4th Qtr

FIG. 1A

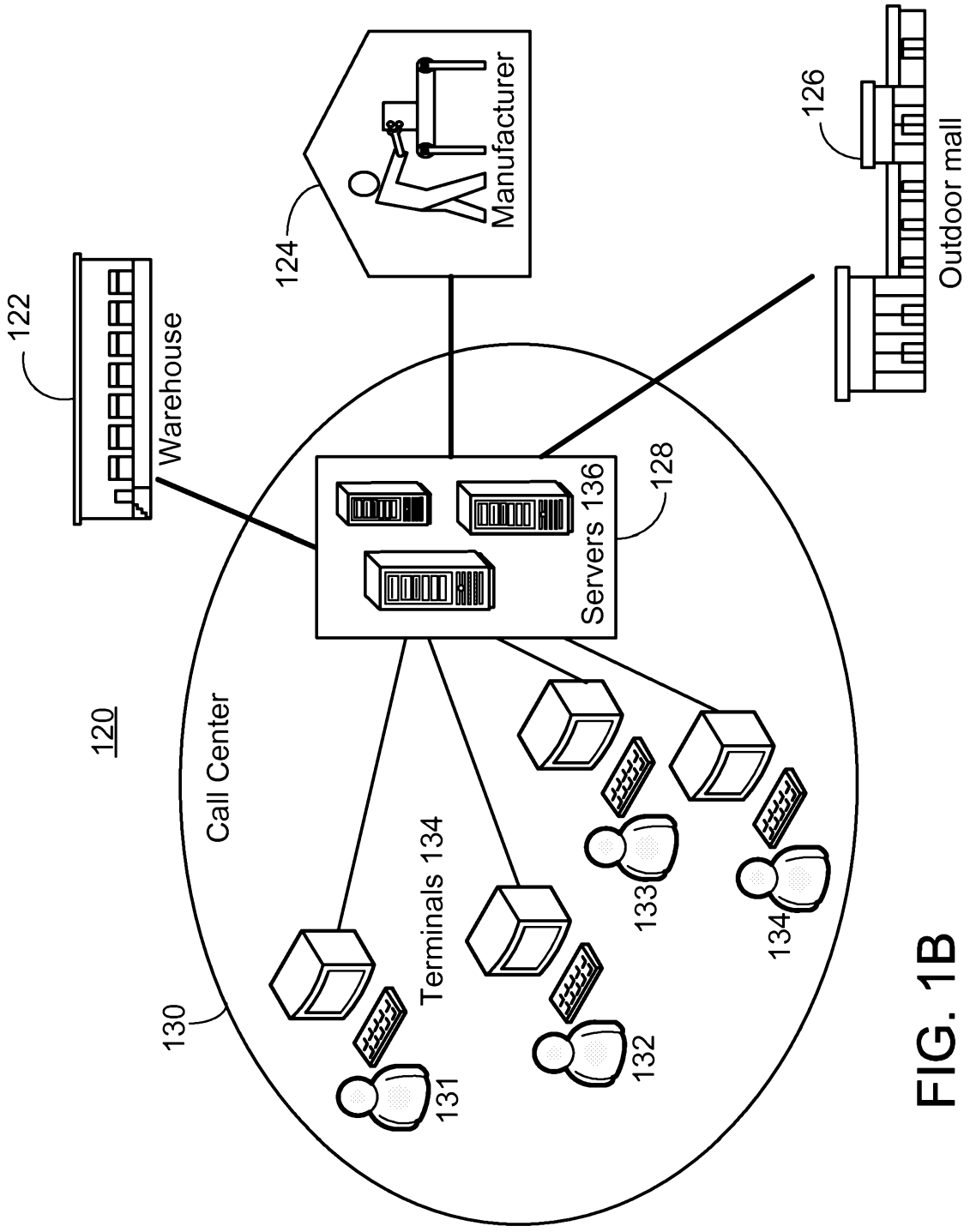
120
Call Center
130
Terminals 134
131
132
133
134
Servers 136
128
Warehouse
122
Manufacturer
124
Outdoor mall
126

FIG. 1B

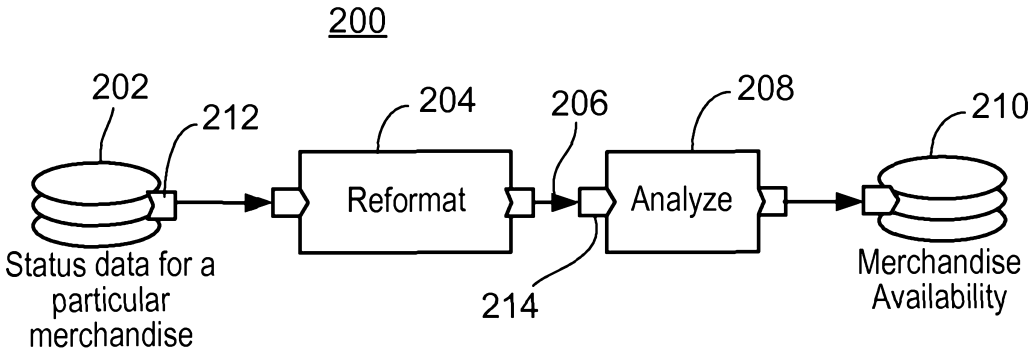
200
202
212
204
206
208
210
Reformat
Analyze
Status data for a particular merchandise
214
Merchandise Availability

FIG. 2

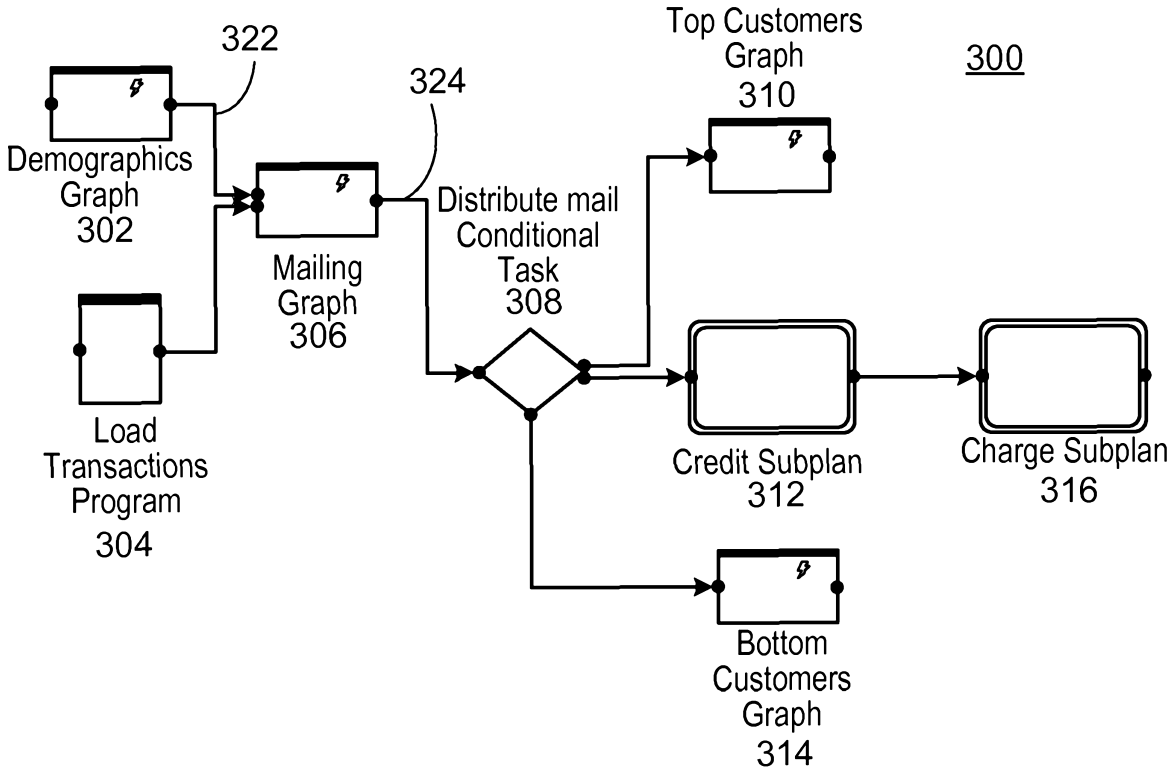
322
324
Top Customers Graph 310
300
Demographics Graph 302
Mailing Graph 306
Distribute mail Conditional Task 308
Load Transactions Program 304
Credit Subplan 312
Charge Subplan 316
Bottom Customers Graph 314

FIG. 3

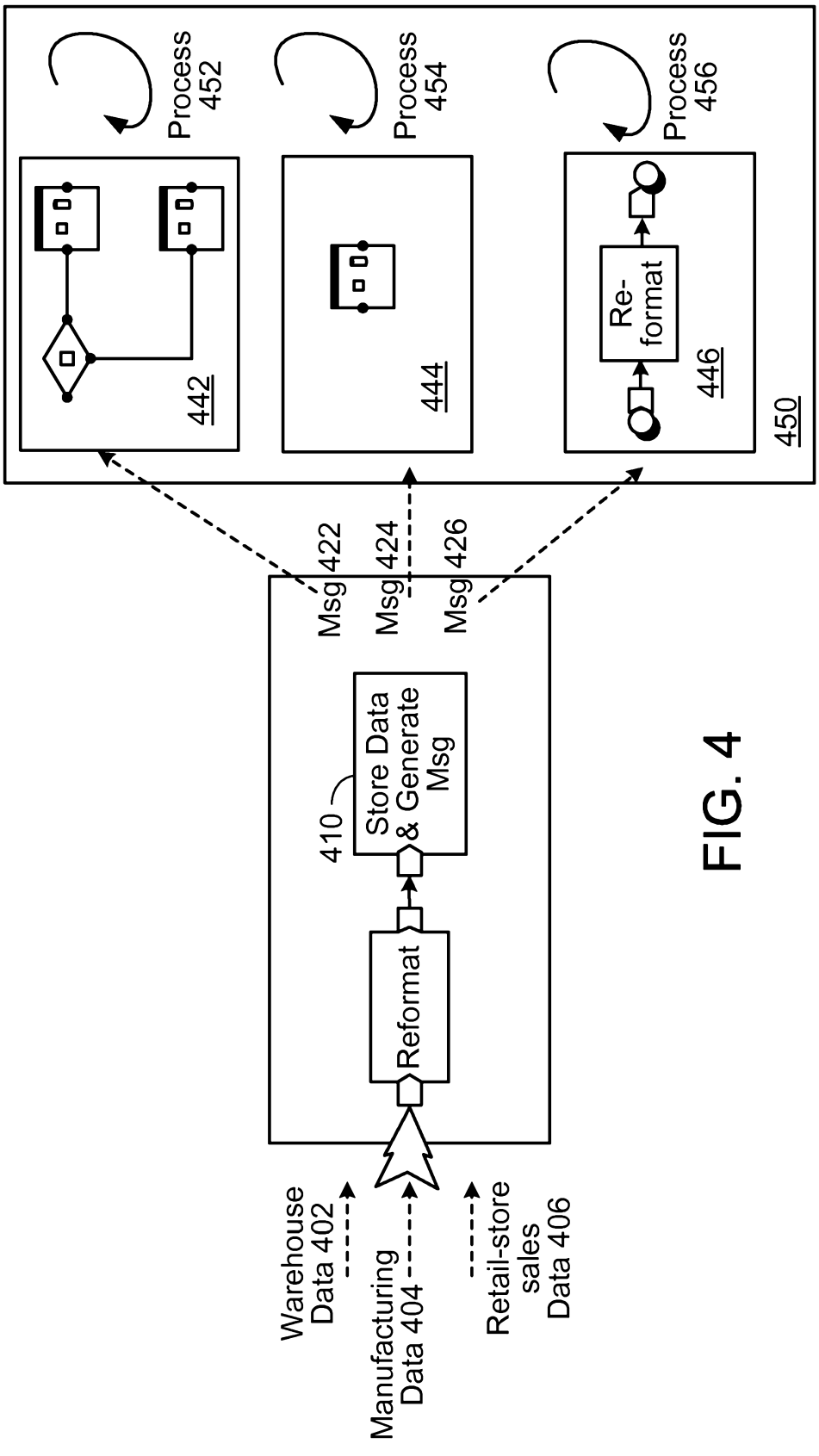
Process 452
Process 454
Process 456
442
444
446
Re-format
450
Msg 422
Msg 424
Msg 426
410
Store Data & Generate Msg
Reformat
Warehouse Data 402
Manufacturing Data 404
Retail-store sales Data 406

FIG. 4

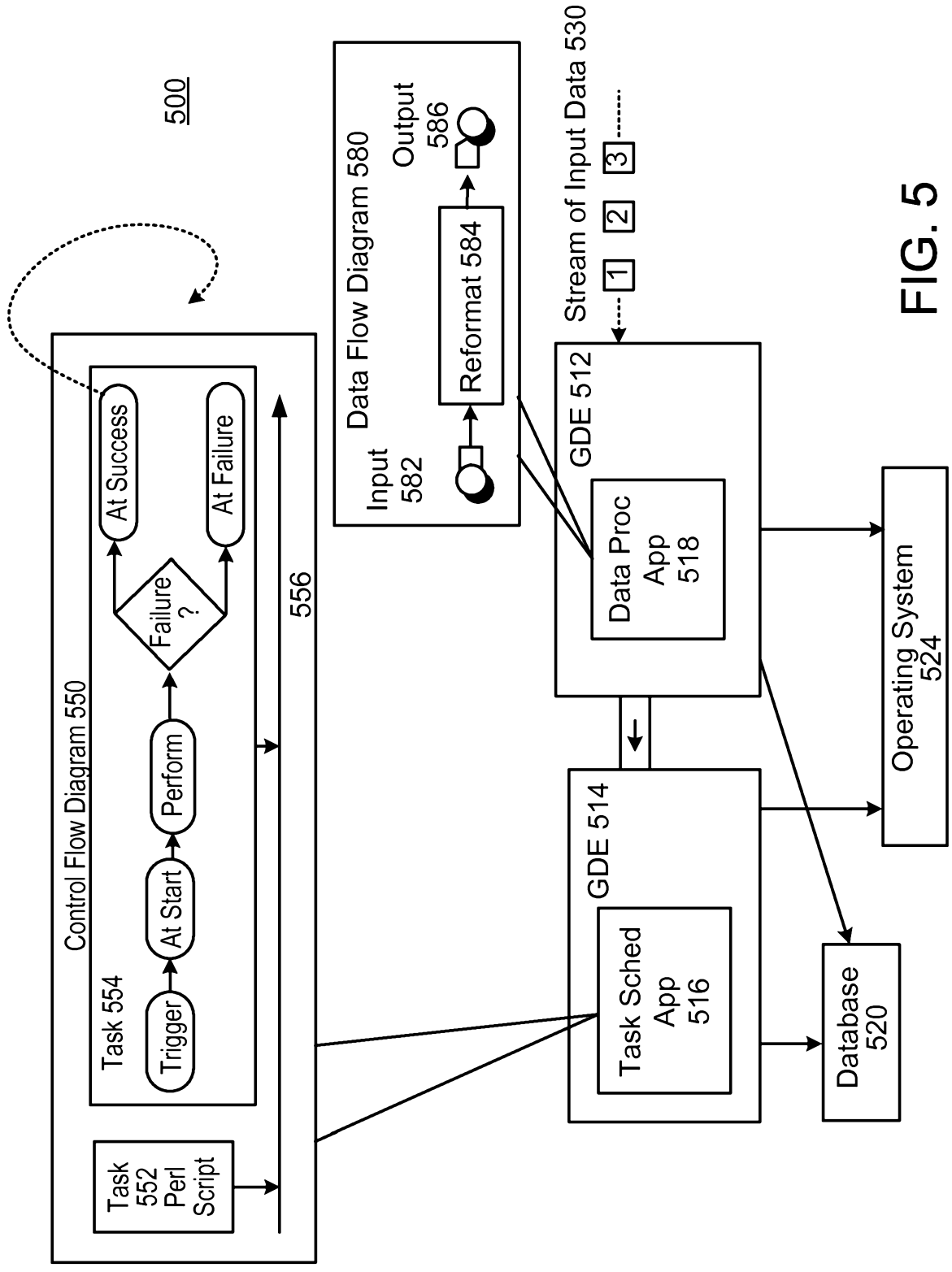
500
Control Flow Diagram 550
Task 552
Perl
Script
Task 554
Trigger
At Start
Perform
Failure
?
At Success
At Failure
556
Data Flow Diagram 580
Input
582
Output
586
Reformat 584
Stream of Input Data 530
1
2
3
GDE 512
Data Proc
App
518
GDE 514
Task Sched
App
516
Operating System
524
Database
520

FIG. 5

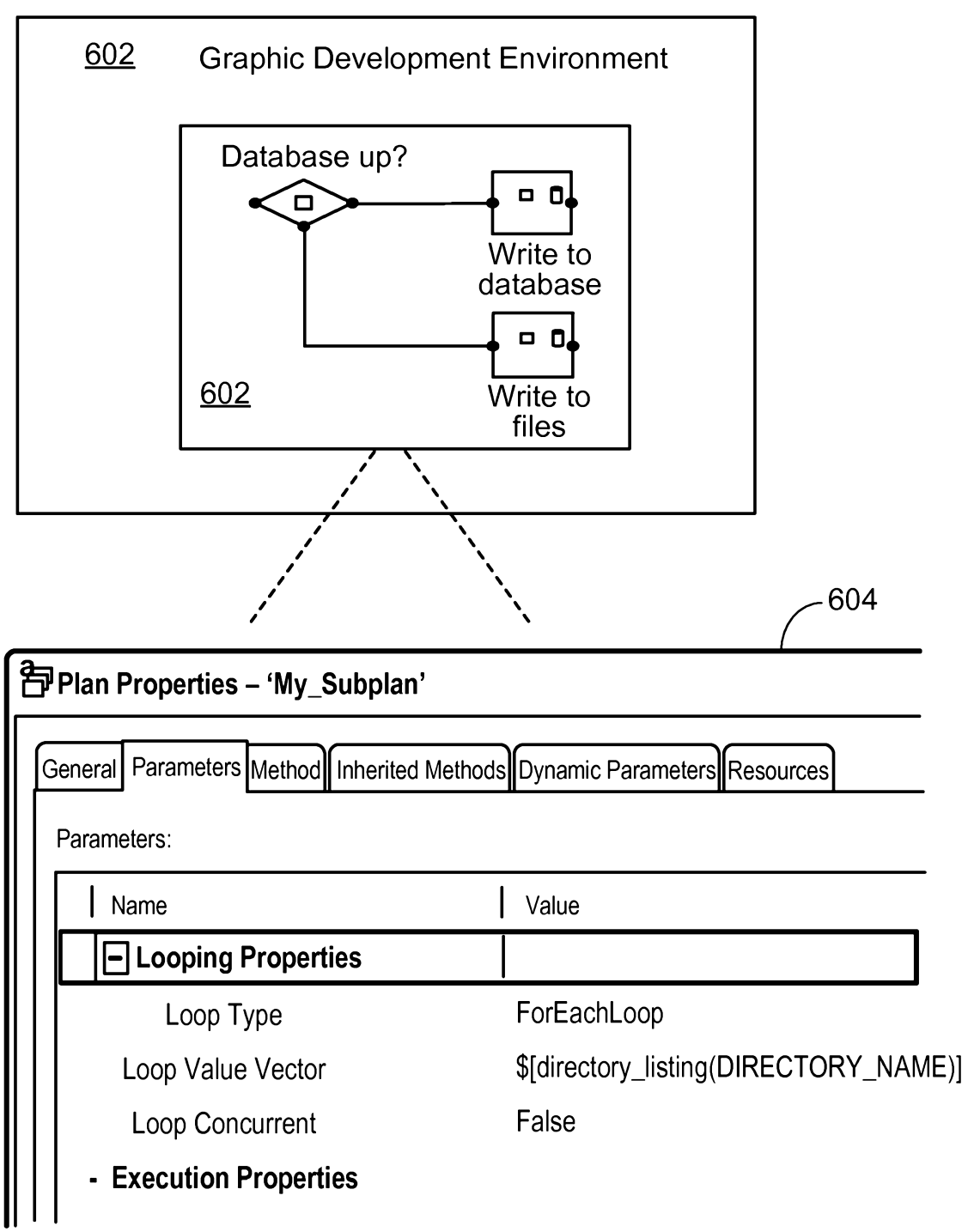
602
Graphic Development Environment
Database up?
Write to database
602
Write to files
604
Plan Properties – 'My_Subplan'
General
Parameters
Method
Inherited Methods
Dynamic Parameters
Resources
Parameters:
Name
Value
Looping Properties
Loop Type
ForEachLoop
Loop Value Vector
$[directory_listing(DIRECTORY_NAME)]
Loop Concurrent
False
- Execution Properties

FIG. 6

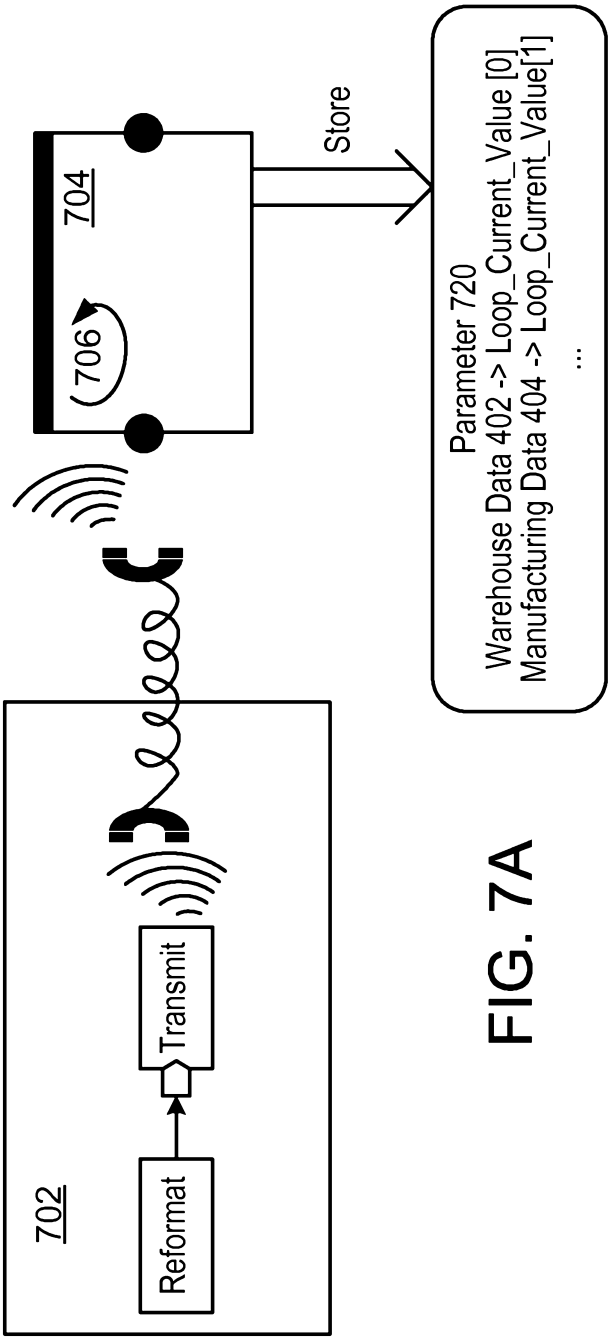
702
Reformat
Transmit
704
706
Store
Parameter 720
Warehouse Data 402 -> Loop_Current_Value [0]
Manufacturing Data 404 -> Loop_Current_Value[1]
...

FIG. 7A

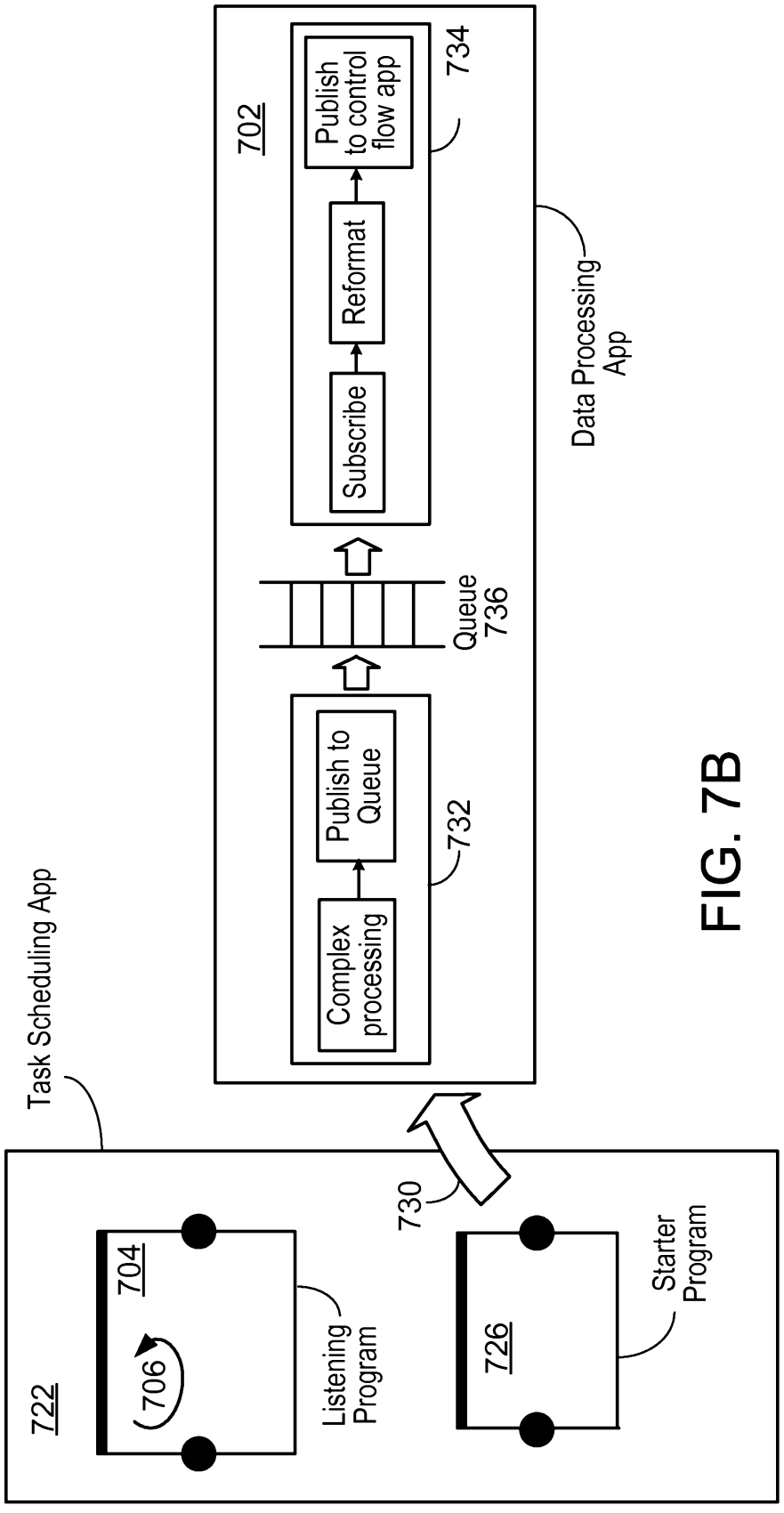
Task Scheduling App
722
704
706
Listening Program
730
726
Starter Program
702
Complex processing
Publish to Queue
732
Queue 736
Subscribe
Reformat
Publish to control flow app
734
Data Processing App

FIG. 7B

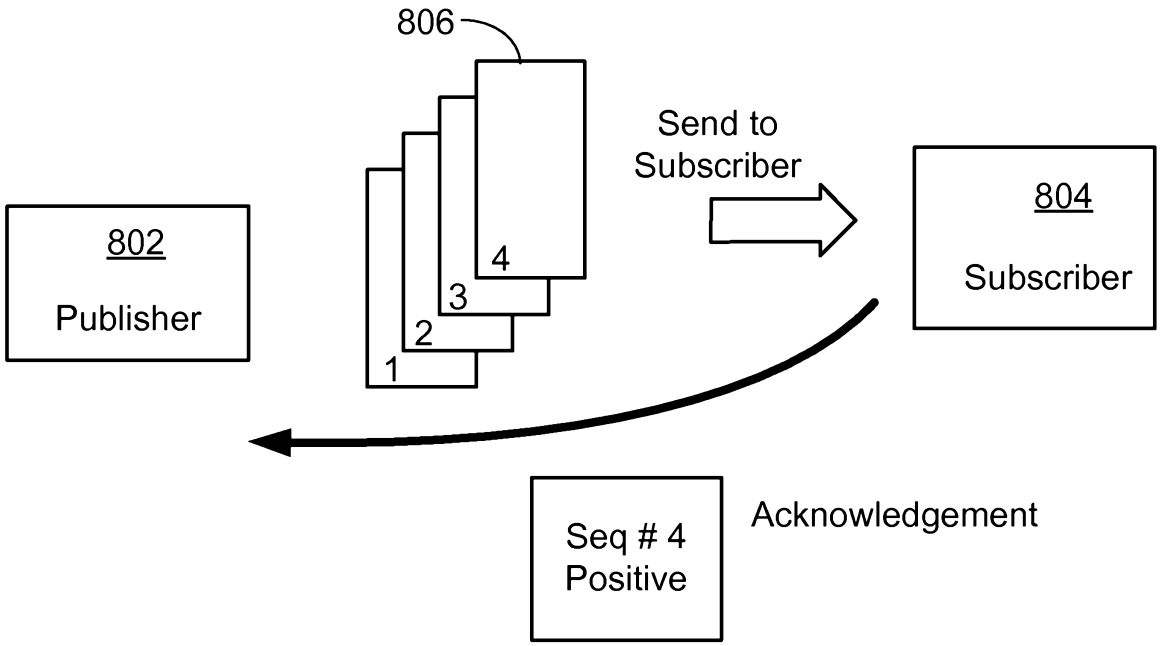
806
Send to
Subscriber
802
Publisher
804
Subscriber
4
3
2
1
Seq # 4
Positive
Acknowledgement

FIG. 8