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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 October 2007 (25.10.2007)

PCT

(10) International Publication Number
WO 2007/120391 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/US2007/004642

(22) International Filing Date:
21 February 2007 (21.02.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/393,093 30 March 2006 (30.03.2006) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

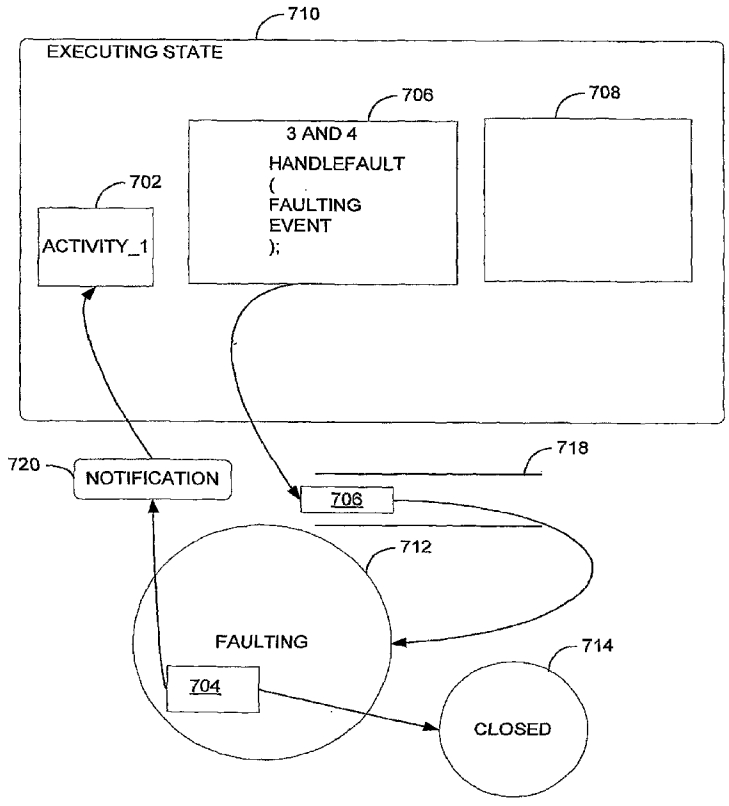
(72) Inventors: **SHUKLA, Dharma**; One Microsoft Way, Redmond, WA 98052-6399 (US). **SCHMIDT, Bob**; One Microsoft Way, Redmond, WA 98052-6399 (US). **MEHTA, Mayank**; One Microsoft Way, Redmond, WA 98052-6399 (US). **TALBERT, Nathan**; One Microsoft Way, Redmond, WA 98052-6399 (US). **SAGAR, Akash, J.**; One Microsoft Way, Redmond, WA 98052-6399 (US). **RAMAN, Karthik**; One Microsoft Way, Redmond, WA 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, 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, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: ASYNCHRONOUS FAULT HANDLING IN PROCESS-CENTRIC PROGRAMS



(57) Abstract: Asynchronous fault handling for a workflow. A state automaton for an activity in the workflow is defined. The state automaton includes at least an executing state, a faulting state, and a closed state and classifies an execution lifetime of the activity. The activity is defined to include work items and includes an execution hierarchy for the work items. Each work item includes an operation for executing a portion of the activity. Each work item is transitioned to the executing state. The included operation of transitioned work items is executed in the executing state. One or more of the transitioned work items are identified in response to the faulting event as a function of the execution hierarchy and the included operation. The faulting event is asynchronously handled by transitioning the one or more identified work items to the faulting state while executing the included operation of the remaining transitioned work items.

WO 2007/120391 A1

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report*

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

ASYNCHRONOUS FAULT HANDLING IN PROCESS-CENTRIC PROGRAMS

BACKGROUND

[0001] Process-oriented or process-centric programs have evolved to enable processing of complex instructions modeling real-world events. Process-centric programs mirror real-world processes and mirror interactions between real-world entities. Existing systems attempt to map business problems to high-level workflows by modeling the business problem. However, real world workflows vary in a variety of dimensions such as (a) execution and modeling complexity, (b) knowledge of the structure of the flow at design time, (c) statically defined or ad-hoc/dynamic, (d) ease of authoring and editing the flow at various points in its lifecycle, and (e) weak or strong association of business logic with the core workflow process. Existing models fail to accommodate all these factors.

[0002] Further, most existing workflow models are based on either language-based approaches (e.g., BPEL4WS, XLANG/S, and WSFL) or application based approaches. Language based approaches are high-level workflow languages with a closed set of pre-defined constructs which help model the workflow process to the user/programmer. The workflow languages carry all of the semantic information for the closed set of constructs to enable the user to build a workflow model. However, the languages are not extensible by the developers and represent a closed set of primitives that constitute the workflow model. The languages are tied to the language compiler shipped by the workflow system vendor. Only the workflow system product vendor may extend the model by extending the language with a new set of constructs in a future version of the product. This often requires upgrading the compiler associated with the language. In addition, the languages usually do not declaratively expose or define functions or operations that can be readily and efficiently used by other programs.

[0003] Application based approaches are applications which have the workflow capabilities within the application to solve a domain specific problem. These applications are not truly extensible nor do they have a programmable model.

[0004] In addition, with the existing approaches, the issues of complexity, foreknowledge, dynamic workflows, authoring ease, and strength of associations with business logic and core workflows are not adequately addressed. There are no extensible, customizable, and re-hostable workflow designer frameworks available to build visual workflow designers to model different classes of workflows. Existing systems lack a rapid application development (RAD) style workflow design experience which allows users to graphically design the workflow process and associate the business logic in a programming language of developer's choice.

[0005] Also, workflow processes deal with cross cutting orthogonal and tangled concerns that span multiple steps of a workflow process model. For example, while parts of the workflow process are designed to participate in long running transactions, other parts of the same process are designed for concurrent execution or for accessing a shared resource. Due to design shortcomings, existing systems fail to provide interleaving of execution threads which enable users to design synchronous or interleaved execution of activities. Still other portions of the same workflow process require tracking, while other portions handle business or application level exceptions. There is a need to apply certain behaviors to one or more portions of a workflow process.

[0006] Some workflow modeling approaches are impractical as they require a complete flow-based description of an entire business process including all exceptions and human interventions. Some of these approaches provide additional functionality as exceptions arise, while other approaches exclusively employ a constraint-based approach instead of a flow-based approach to modeling a business process. Existing systems implement either

the flow-based or constraint-based approach. Such systems are too inflexible to model many common business situations. These systems also lack the capability to asynchronously handle exceptions or cancellations.

SUMMARY

[0007] Embodiments of the invention enable asynchronous fault or exception handling by having a faulting state in a state automaton defining execution lifetime of an activity in the workflow. By having the faulting state, aspects of the invention enable developers or programs to declaratively design programs for exception or fault handling such that portions of the program or the activity may be in fault handling in the faulting state while other portions of the program or the activity may be unaffected by the exception or the faulting event.

[0008] Alternative embodiments of the invention enable propagation or transmission of a notification of fault handling. In yet another alternative embodiment, such propagation or transmission of the notification may be suppressed or inhibited. In addition, a further alternative embodiment responds to input from a user for handling post-faulting or post-exception operations.

[0009] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

[0010] Other features will be in part apparent and in part pointed out hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] FIG. 1 is a block diagram illustrating an existing programming paradigm.

[0012] FIG. 2 is an exemplary block diagram illustrating a virtualization of a workflow design framework according to an embodiment of the invention.

[0013] FIG. 3 is an exemplary diagram illustrating an exemplary workflow according to an embodiment of the invention.

[0014] FIG. 4 is a diagram illustrating an exemplary computing environment of a system for processing workflow activities according to an embodiment of the invention.

[0015] FIG. 5 is a diagram illustrating a hierarchical structure of a workflow activity according to an embodiment of the invention.

[0016] FIG. 6 is a diagram illustrating an exemplary state automaton describing execution lifetime of an activity according to an embodiment of the invention.

[0017] FIGS. 7A to 7E are block diagrams illustrating an asynchronous handling of faulting events of a workflow according to an embodiment of the invention.

[0018] FIG. 8 is a flow diagram illustrating a method for asynchronously handling of a faulting event for an activity of a workflow according to an embodiment of the invention.

[0019] FIG. 9 is a block diagram illustrating an exemplary computer-readable medium on which aspects of the invention may be stored.

[0020] Appendix A illustrates an exemplary implementation of declaratively raising of an exception according to an embodiment of the invention.

[0021] Corresponding reference characters indicate corresponding parts throughout the drawings.

DETAILED DESCRIPTION

[0022] Referring first to FIG. 1, a block diagram illustrates an existing programming paradigm for designing programs for process-centric activities, such as a workflow. For example, the diagram shows a three-level virtualization model of existing program

paradigm with a level of a managed execution environment being the highest level and a processing unit being the lowest level. In this programming design system, even at the managed execution environment level, programs, especially process-centric programs handling workflow processes, lack the ability and efficiency to accommodate complex interactions between processes in a workflow.

[0023] It is known by those skilled in the art that certain constraints are associated with designing software or application programs. In this example, in writing an operating system software program 104, the programming codes or routines are dependent on the type or configuration of processing units 102, being specific to the type of computing architecture (e.g., IBM® compatible, APPLE® computers, or other systems), or other constraints. In addition, programming languages typically need to accurately identify and utilize data structures such as stacks, heap, thread base, or other hardware-specific structures for the operating system 104 to function properly.

[0024] In dealing with complex workflow processes, existing applications use a concept of a managed execution environment 106 (e.g., a runtime environment where programs may share functions or common object-oriented classes) in which programs written one programming language may call functions in other programs written in a different programming language. In such execution environment, these programs in different programming languages are compiled to an intermediate language such that the managed execution environment 106 may expose parameters, arguments, or schemas or functions to the different programs so that the programs may interact with one another.

[0025] While this execution environment 106 creates a common communication environment between programs, the execution environment 106 includes various strict requirements that may not be suitable for handling the complexity and capability of process-centric programs. For example, the execution environment 106 requires programs

be confirmed to a specific file format. The execution environment 106 also requires that functions or operations in the programs use a fixed set of functions or a class of functions defined by the execution environment 106.

[0026] Embodiments of the invention build on an extensible foundation or framework 202 in FIG. 2 to overcome the shortcomings of existing programming model. By allowing programs written in any programming language and composed in any file format, aspects of the invention enable program developers to design programs with specific functions without compromising its functionalities and specifics. By defining activities, such as workflow tasks or processes, as the base class to be executed in the workflow framework, developers can easily and efficiently build domain specific (e.g., specific execution environments such as programs in the healthcare industrial, financial industry, or the like) operation codes (hereinafter "op-code") without adhering to the rigid, hard-coded, inflexible, and the fixed set of functions or activities classes in the existing execution environment. In addition, the workflow foundation embodying aspects of the invention is a continuation based runtime layered on top of any existing framework (e.g., either a managed execution environment, operating system environment, or hardware processing unit level).

[0027] Aspects of the invention free the constraint of defining activities in a particular file format by enabling workflow designs in any fashion or representation (e.g., a flow chart, a diagram, a numbered description, or the like) as long as activities in the workflow can be constructed from the representation of the workflow designs.

[0028] In addition, the workflow framework or foundation is able to handle fault or exception raised from a lower level (e.g., OS) or exception raising functions written in other formats (e.g., intermediate language).

[0029] FIG. 3 illustrates a simplistic view of a workflow 300 according to an embodiment of the invention. For example, the workflow 300 may be a workflow for processing a purchase order, and this purchase order workflow 300 may include processes or activities such as receive a purchase order, send confirmation to a customer, approve the purchase order by a manager, or the like.

[0030] The workflow 300 may start from a starting point 302. For example, the starting point 302 for a purchase-order workflow may be receiving an order from a customer. The workflow 300 may also include a conditional statement 304 (such as an "IF statement" or a "WHILE statement"), and it can be subdivided into additional conditional statements 306 and 308. The workflow 300 may also include a parallel structure 310, which further includes one or more sequences or activities 312. For example, the parallel structure 310 includes activities, such as checking the inventory and updating the available shippers, be processed in parallel. In the example shown, activities such as "Send E-mail" and "Get Approval" may be processed in parallel. At "drop activities here" 316, a user may further add or supplement more activities into the workflow 300. To complete the workflow 300, the processes or activities will conclude in a complete step or point 314.

[0031] In one embodiment, the activities may be arranged hierarchically in a tree structure (see FIG. 5) 500 or other execution sequences. For example, an activity may be a composite activity in which the activity includes more than one work item associated therewith. In another embodiment, a collection of activities may be a composite activity. An activity method or operation may be in a root node 502 with two children or leaf nodes 504 and 506. The activity methods or operations in the children nodes 504 and 506 (e.g., work item_1 and work item_2, respectively) may be executed according to the hierarchical structure. In addition, the children nodes 504 and 506 may also include other children nodes having respective work items to be executed.

[0032] In another embodiment, activities include one or more of the following types: a simple activity, container activity and root activity. In this embodiment, there is one root activity in the model, and none or any quantity of simple activities or container activities inside the root activity. A container activity may include simple or container activities. The entire workflow process may be used as an activity to build higher-order workflow processes. Further, an activity may be interruptible or non-interruptible. A non-interruptible composite activity does not include interruptible activities. A non-interruptible activity lacks services that would cause the activity to block.

[0033] Moreover, in executing activities and the work items included in the activities, the workflow framework or an execution context or environment defines a scope or boundary for each of the work items. This scope or boundary includes and exposes information (e.g., in the form of data, metadata, or the like) such as the shared data or resources to be accessed by the work items, associated properties, handlers, constraints and interactions between autonomous agents. Also, each activity may be configured by a user code in any programming language. For example, the user code may represent business or application logic or rules written in a specific domain or execution environment. Each activity may support pre-interception hooks and post-interception hooks into execution in the user code. Each activity has associated runtime execution semantics and behavior (e.g., state management, transactions, event handling and exception handling). Activities may share state or resources with other activities. In addition, activities may be primitive activities or grouped into a composite activity. A primitive or basic activity has no substructure (e.g., child activities), and thus is a leaf node in a tree structure. A composite activity contains substructure (e.g., it is the parent of one or more child activities).

[0034] FIG. 4 is a diagram illustrating a system 400 for processing workflow activities according to an embodiment of the invention. The system 400 includes a processor 402, which may be a processing unit or a collection of processing units. The system 400 also includes a storage or memory area 404 for storing data accessible by the processor 402. In one embodiment, the system 400 may be a computer having one or more processors or processing units (e.g., processor 402) and a system memory (e.g., memory area 404) having other components to couple various system components including the system memory to the processor 402.

[0035] In one example, the memory area 404 may include computer readable media, either volatile, nonvolatile, removable, or non-removable media, implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. For example, computer storage media include RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium that may be used to store the desired information and that may be accessed by the system 400. The memory 404 may also include communication media embodying computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and include any information delivery media. Those skilled in the art are familiar with the modulated data signal, which has one or more of its characteristics set or changed in such a manner as to encode information in the signal. Wired media, such as a wired network or direct-wired connection, and wireless media, such as acoustic, RF, infrared, and other wireless media, are examples of communication media. Combinations of any of the above are also included within the scope of computer readable media.

[0036] For example, the memory area 404 stores a plurality of activities 406 for processing in a workflow (e.g., the workflow 300). Each of the plurality of activities 406 includes one or more work items, and the work items may be organized in a hierarchical structure such as a tree structure (see FIG. 5). In processing the plurality of activities 406, the processor 402 accesses or executes a scheduler 408, which is configured to set up an organized set of activities.

[0037] For example, the processor 408 accesses the work items in the plurality of activities 406 via a component or a set of computer-executable instructions such as the scheduler 408 to enqueue or to store the work items 422 to a queue 410. A dispatcher 412, accessible by the processor 402, dispatches the work items 422 for execution. For example, a work item 422-1 may include an activity method or an activity operation 424, routine, or a collection of codes for performing a function of "requesting input from a user". One or more other activity methods, activity operations, routines, or codes may be included in each of the work items 422 without departing from the scope of the invention.

[0038] Once the work items 422 are dispatched by the dispatcher 412, the processor 402 executes each of the methods 424 in the work items 422 at 414. In the example of work item 422-1, the processor 402 may provide a user via a user interface (UI) to input the requested information or data. In another embodiment, the processor 402 may connect to or access an external data source for requesting input from the user. Upon completion of the activity method or activity operation 424, the processor 402 concludes execution of the work items 422 at 416. In one embodiment, the processor 402 passivates the executing state of work items at 418 to a data store 420.

[0039] In another embodiment, the processor 402 executes the work items 422 according to a state automaton, such as the automaton shown in FIG. 6, which is a diagram illustrating an exemplary state automaton 600 describing processing states of work items

associated with an activity according to an embodiment of the invention. In one embodiment, the state automaton 600 defines an execution lifetime of an activity. In one example, the state automaton 600 may include an initialized state, an executing state, and a closed state (as shown in FIG. 4). In another embodiment, the state automaton 600 includes an initialized state 602, an executing state 604, a canceling state 606, a faulting state 608, a compensating state 610, and a closed state 612.

[0040] For example, the state automaton 600 describes a process flow of execution of work items (e.g., work items 422) in a workflow activity. The work item 422-1, as illustrated in FIG. 4, is first initialized when it is enqueued in the queue 410. The work item 422-1 is next dequeued or removed from the queue 410 to the dispatcher 412 before being executed in the executing state (e.g., the executing state 604 in FIG. 6). Depending on the parameters or conditions during the execution of the work item 422-1, the work item 422-1 may proceed to the canceling state 606 (e.g., the canceling state 426 in FIG. 4) or the faulting state 608. In one embodiment, the work item 422-1 may proceed from the canceling state 606 to the faulting state 608. In an alternative embodiment, the compensating state 610 describes a set of operations or functions to be performed when faulting or exception has occurred.

[0041] For example, suppose an exception occurs during the execution of a work item (e.g., work item 422-1), such as a parameter for a function is missing. The system 400 transitions the work item 422-1 to the faulting state 608. In doing so, the system 400 also performs garbage collection (e.g., removing previously executed portion of the operations from cache or memory, reset parameter values, or the like) operations in the compensating state 610 before transitioning the work item 422-1 to the closed state 612. For example, work items in the compensating state 610 may trigger operations such as recovering data

that was previously used for executing other work items. The closed state 612 indicates that the execution of the activity (e.g., activity 500 in FIG. 5) has completed.

[0042] In one embodiment, the state automaton 600 establishes relationship between work items in a composite activity. For example, one of the relationship rules may include that, before transitioning to the closed state 612 methods or work items in the root node of the activity tree, all of the work items in the children nodes should be in the initialized state 602 or the closed state 612. Another rule may require that, in order to transition the work items in the children node of the activity tree to the executing state 604, the work item in the root node must already be in the executing state 604.

[0043] In another embodiment, one or more additional states may be defined in the state automaton 600 without departing from the scope of embodiments of the invention.

[0044] Referring next to FIGS. 7A and 7E, block diagrams illustrate asynchronous handling of faulting events in a workflow according to an embodiment of the invention. For simplistic purposes only and without limitations, FIG. 7A shows a composite activity 702 which includes three children work items organized in a tree structure: transaction_1 704, transaction_2 706, and transaction_3 708. As illustrated, the root activity 702 includes a method to "write text on display." Activity methods or operations for the work items above also include the following:

```
transaction_1 704:
{ INSERT TEXT ("1 AND 2");
  HANDLEFAULT();
}
transaction_2 706:
{ INSERT TEXT ("3 AND 4");
  HANDLEFAULT();
```

```
}
transaction_3 708:
{ INSERT TEXT("5 AND 6");
  PAUSE 180 SECONDS;
  INSERT TEXT ("END");
}
```

[0045] In FIG. 7B, the transaction_1 704, the transaction_2 706, and the transaction_3 708 are transitioned to the executing state 710. As illustrated, the transaction_1 704 executes the included operations by inserting texts ("1 and 2") on the display (e.g., a user interface 428) to a user 430.

[0046] While in the executing state 710, a faulting event 722 or an exception has occurred. The faulting event 722 may include a warning notification for missing data, an execution fault, an inaccurate access to a data store, or the like. In this example, the transaction_1 704 includes a handleFault() function 716 for handling the faulting event 722. In one embodiment, the handleFault function 716 resembles a "catch" function for fault handling in other execution environments, such as an operating system or a managed execution environment. As such, the fault propagation or dispatch to the handleFault function 716 or the "catch" handler is asynchronous.

[0047] Upon the occurrence of the faulting event 722, the transaction_1 704 transitions to a faulting state 712, and the transaction_1 704 is transitioned to a closed state 714. In one embodiment, in responding to the faulting event 722, the handleFault() function 716 is called and is placed in a queue (not shown) for processing.

[0048] With this well-defined protocol for exception propagation and handling, alternative embodiments may handle multiple exceptions, and multiple exceptions may be

scheduled while the propagation of exceptions may be interleaved with the normal program execution.

[0049] In FIG. 7C, the transaction_2 706 and the transaction 708 are in the executing state 710. Similar to executing the transaction_1 704, the transaction_2 706 executes the included operations. In this example, the texts ("3 and 4") are inserted on the display. In addition, the transaction_2 706 also includes a similar handleFault() function as the handleFault() function 716 in the transaction_1 704 for handling the faulting event 722.

[0050] In an alternative embodiment, the handleFault() function 716 may propagate or transmit a notification 720 to the remaining work items in the executing state 710 as a function of the execution hierarchy or the execution hierarchical structure of the activity. For example, while the transaction_1 704 is in faulting state 712, the handleFault() function 716 may propagate the notification 720 (e.g., a "throw" function) so that the handleFault() function of the parent Activity_1 702 may handle it as if the notification 720 is a faulting event or an exception. In one embodiment, a child activity may limit the target of the throw function to its parent in the activity tree. In another embodiment, exception handling may be highly associated with or tied to the tree like structure of activities.

[0051] By establishing the faulting state 712 for handling faulting events, embodiments of the invention enable asynchronous faulting handling or exception handling, and the remaining work items or activities in the executing state 710 continue to be executed. In addition, another alternative embodiment enables scheduling of handling faulting events. For example, upon responding to the notification 720, the transaction_2 706 may be placed in a scheduler queue 718 before being transitioned to the faulting state 712. In another embodiment, the notification 720 may be suppressed such that other work items or activities in the executing state 710 continue to be executed. In one embodiment, the

transaction_1 704 transitions to a closed state 714 after propagating or transmitting the notification 720. In yet another embodiment, fault propagation and handling survive and span across passivation cycles.

[0052] In FIG. 7D, the transaction_3 708 is being executed in the executing state 710. For example, the included operations of the transaction_3 708 insert the texts "(5 and 6") and pause 180 seconds before inserting the text ("END") on the display. The included operations, however, do not include functions for faulting handling. As such, upon completion of the included operations, the transaction_3 708 is transitioned to the closed state 714 in FIG. 7E. In addition, the transaction_2 706 also transitions to the closed state 714 after being dequeued from the scheduler queue 718 to the faulting state 712.

[0053] Without limitations, Appendix A illustrates an exemplary implementation of declaratively raising of an exception according to an embodiment of the invention. In one embodiment, programmers or developers may design a fault handler for handling a particular type of faulting events or exceptions. In yet another embodiment, work items or activities in the workflow may not include a function or incapable to handle faulting events. In this embodiment, the workflow execution environment handles the faulting events. In yet another embodiment, one or more post-fault-handling operations may be provided to the user via the UI 428 to the user 430 in FIG. 4.

[0054] While FIGS. 7A to 7E illustrate snapshots of the executing state or parts of the state automaton sequentially (e.g., transactions are executed sequentially), work items in the executing state may be processed simultaneously or substantially simultaneously without departing from the scope of the invention.

[0055] FIG. 8 is a flow diagram illustrating a method for asynchronously handling of a faulting event for an activity of a workflow according to an embodiment of the invention. For example, the method illustrated in FIG. 8 may be represented as computer-executable

instructions to be stored in a computer-readable medium as shown in FIG. 9. For example, a state machine 902 defines a state automaton (e.g., state automaton 600) for an activity at 802, and the state automaton includes at least an executing state, a faulting state, and a closed state. An activity component 904 defines the activity to include a plurality of work items at 804. The defined activity has an execution hierarchy or an execution sequence (e.g., a tree structure) for the plurality of work items. Each of the work items including an operation for executing a portion of the activity.

[0056] A scheduler component 906 transitions each of the work items to the executing state at 806. An execution component 908 executes the included operation of transitioned work items in the executing state at 808. At 810, an identification component 910 identifies one or more of the transitioned work items in response to the faulting event based on the execution hierarchy and the included operation. At 812, a fault handler 912 asynchronously handles the faulting event by invoking a fault handling operation (e.g., the handleFault() function 716) in the one or more identified work items to transition the one or more identified work items to the faulting state while executing the included operation of the remaining transitioned work items not identified in response to the faulting event by the identification component. In one embodiment, the fault handler 912 asynchronously handles the faulting event by transitioning the one or more identified work items to the faulting state. In yet another embodiment, the fault handler 912 asynchronously handles the faulting event by enqueuing the one or more identified work items in a scheduler queue (e.g., scheduler queue 718).

[0057] In an alternative embodiment, the computer-readable medium 900 further includes a fault propagation component 914 for transmitting a notification from the one or more identified work items to the remaining transitioned work items as a function of the execution hierarchy of the activity. The notification 720 indicates that the identified one

or more work items are in the faulting state. In a further embodiment, the computer-readable medium 900 further includes a transition component 916 for transitioning the remaining transitioned work items from the executing state to the faulting state in response to the transmitted notification.

[0058] The computer-readable medium may also include a compensation component 918 for recovering or compensating data associated with the activity as a function of the asynchronously handling the faulting event in yet another alternative embodiment. An inhibition component may also be part of the computer-readable medium 900 for suppressing the transmission of the notification to the remaining transitioned work items.

[0059] Although described in connection with an exemplary computing system environment, such as the system 400 in FIG. 4, embodiments of the invention are operational with numerous other general purpose or special purpose computing system environments or configurations. The computing system environment is not intended to suggest any limitation as to the scope of use or functionality of any aspect of the invention. Moreover, the computing system environment should not be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with aspects of the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, mobile telephones, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0060] Embodiments of the invention may be described in the general context of computer-executable instructions, such as program modules, executed by one or more

computers or other devices. Generally, program modules include, but are not limited to, routines, programs, objects, components, and data structures that perform particular tasks or implement particular abstract data types. Aspects of the invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

[0061] In operation, the system 400 executes computer-executable instructions such as those illustrated in the figures, such as FIG. 8, to implement aspects of the invention.

[0062] The order of execution or performance of the operations in embodiments of the invention illustrated and described herein is not essential, unless otherwise specified. That is, the operations may be performed in any order, unless otherwise specified, and embodiments of the invention may include additional or fewer operations than those disclosed herein. For example, it is contemplated that executing or performing a particular operation before, contemporaneously with, or after another operation is within the scope of aspects of the invention.

[0063] Embodiments of the invention may be implemented with computer-executable instructions. The computer-executable instructions may be organized into one or more computer-executable components or modules. Aspects of the invention may be implemented with any number and organization of such components or modules. For example, aspects of the invention are not limited to the specific computer-executable instructions or the specific components or modules illustrated in the figures and described herein. Other embodiments of the invention may include different computer-executable instructions or components having more or less functionality than illustrated and described herein.

[0064] When introducing elements of aspects of the invention or the embodiments thereof, the articles "a," "an," "the," and "said" are intended to mean that there are one or more of the elements. The terms "comprising," "including," and "having" are intended to be inclusive and mean that there may be additional elements other than the listed elements.

[0065] Having described aspects of the invention in detail, it will be apparent that modifications and variations are possible without departing from the scope of aspects of the invention as defined in the appended claims. As various changes could be made in the above constructions, products, and methods without departing from the scope of aspects of the invention, it is intended that all matter contained in the above description and shown in the accompanying drawings shall be interpreted as illustrative and not in a limiting sense.

APPENDIX A

[0066] <myActivities:Sequence x:Name="myWorkflow"
x:Class="myApp.myWorkflow"
xmlns:myActivities="http://schemas.com/myActivities"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
<myActivities:WriteLine Text="One"/>
<myActivities:WriteLine Text="Two"/>
<ThrowActivity FaultType="{x:Type InvalidOperationException}"/>
<myActivities:WriteLine Text="Unreachable code"/>
<FaultHandlersActivity>
<FaultHandlerActivity FaultType="{x:Type
InvalidOperationException}">
<myActivities:WriteLine Text="Three"/>
</FaultHandlerActivity>
<FaultHandlerActivity FaultType="{x:Type
AppDomainUnloadedException}">
<myActivities:WriteLine Text="Four"/>
</FaultHandlerActivity>
</FaultHandlersActivity>
</myActivities:Sequence>
<myActivities:Sequence x:Name="myWorkflow" x:Class="myApp.myWorkflow"
xmlns:myActivities="http://schemas.com/myActivities"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">

```
        <myActivities:WriteLine Text="Hello World"/>
        <ThrowActivity FaultType="{x:Type InvalidOperationException}" />
        <myActivities:WriteLine Text="Unreachable Code"/>
    </myActivities:Sequence>
```

CLAIMS

What is claimed is:

1. A method for asynchronous handling a faulting event (722) for an activity of a workflow, said method comprising:

defining a state automaton (600) for an activity (702), said state automaton (600) including at least an executing state (604), a faulting state (608), and a closed state (612), said state automaton (600) classifying an execution lifetime of the activity (702);

defining the activity (702) to include a plurality of work items (422), said defined activity (702) having an execution hierarchy (500) for the plurality of work items (422), each of the work items (422) including an operation for executing a portion of the activity (702);

transitioning each of the work items (422) to the executing state (604);

executing the included operation of transitioned work items (422) in the executing state (604);

identifying one or more of the transitioned work items in response to the faulting event (722) as a function of the execution hierarchy (500) and the included operation; and

asynchronously handling the faulting event (722) by transitioning the one or more identified work items (422) to the faulting state (608) while executing the included operation of the remaining transitioned work items not identified in response to the faulting event (608).

2. The method of claim 1, wherein asynchronously handling the faulting event (722) comprises invoking a fault handling operation in the one or more identified work items (422).

3. The method of claim 2, wherein invoking the fault handling operation comprises enqueuing the one or more identified work items in a scheduler queue (718).

4. The method of claim 1, further comprising propagating a notification (720) from the one or more identified work items (422) to the remaining transitioned work items (422) as a function of the execution hierarchy (500) of the activity (702), said notification (720) indicating that the identified one or more work items (422) being in the faulting state (608).

5. The method of claim 4, further comprising transitioning the remaining transitioned work items (422) from the executing state (604) to the faulting state (608) in response to the propagated notification (720).

6. The method of claim 4, further comprising suppressing the propagation of the notification (720) to the remaining transitioned work items (422).

7. The method of claim 1, further comprising providing operations to a user for post-faulting handling in response to the one or more identified work items (422) being transitioned to the faulting state (608).

8. The method of claim 1, wherein one or more computer-readable media have computer-executable instructions for performing the method of claim 1.

9. A system (400) for asynchronously handling faulting events (722) in a workflow, said system (400) comprising:

a storage (404) for storing data associated with work items of an activity (406) in the workflow, said activity (406) having an execution hierarchy (500) for the work items (422), each of the work items (422) including an operation for executing a portion of the activity (406);

a processor (402) configured to executing computer-executable instructions for:

defining a state automaton (600) for the activity (406), said state automaton (600) including at least an executing state (604), a faulting state (608), and a closed state (612), said state automaton (600) classifying an execution lifetime of the activity (406);

transitioning each of the work items (422) to the executing state (604);

executing the included operation of transitioned work items (422) in the executing state (604);

identifying one or more of the transitioned work items (422) in response to a faulting event (608) based on the execution hierarchy (500) and the included operation; and

asynchronously handling the faulting event (722) by transitioning the one or more identified work items (422) to the faulting state (608) while executing the included operation of the remaining transitioned work items (422) not identified in response to the faulting event (722).

10. The system (400) of claim 9, wherein the processor (402) is configured to asynchronously handle the faulting event (722) using one or more of the following method: by invoking a fault handling operation in the one or more identified work items (422), and by enqueuing the one or more identified work items (422) in a scheduler queue (718).

11. The system (600) of claim 9, further comprising a component for propagating a notification (720) from the one or more identified work items to the remaining transitioned work items (422) as a function of the execution hierarchy (500) of the activity (406), said notification (720) indicating that the identified one or more work items (422) being in the faulting state (608).

12. The system (400) of claim 11, further comprising a component for transitioning the remaining transitioned work items from the executing state (604) to the faulting state (608) in response to the propagated notification (720).

13. The system (400) of claim 9, further comprising a component for compensating for recovering data associated with the activity from the storage (404) as a function of the asynchronously handling the faulting event (722) by the processor (402).

14. The system (400) of claim 9, further comprising a component for suppressing the propagation of the notification (720) to the remaining transitioned work items (422).

15. One or more computer-readable media (900) having computer-executable components for asynchronously handling a faulting event (722) in a workflow, said computer-executable components comprising:

a state machine (902) for defining a state automaton (600) for an activity (406), said state automaton (600) including at least an executing state (604), a faulting state (608), and a closed state (612), said state automaton (600) classifying an execution lifetime of the activity (406);

an activity component (904) for defining the activity (406) to include a plurality of work items (422), said defined activity (406) having an execution hierarchy (500) for the plurality of work items (422), each of the work items (422) including an operation for executing a portion of the activity (406);

a scheduler component (906) for transitioning each of the work items (422) to the executing state (604);

an execution component (908) for executing the included operation of transitioned work items (422) in the executing state (604);

an identification component (910) for identifying one or more of the transitioned work items in response to the faulting event (722) based on the execution hierarchy (500) and the included operation; and

a fault handler (912) for asynchronously handling the faulting event (722) by invoking a fault handling operation in the one or more identified work items (422) to transition the one or more identified work items (422) to the faulting state (608) while executing the included operation of the remaining transitioned work items (422) not identified in response to the faulting event (722) by the identification component (910).

16. The computer-readable media (900) of claim 15, wherein the fault handler (912) enqueues the one or more identified work items (422) in a scheduler queue (718) to transition the one or more identified work items to the faulting state (608).

17. The computer-readable media (900) of claim 15, further comprising a fault propagation component (914) for transmitting a notification (720) from the one or more identified work items (422) to the remaining transitioned work items (422) as a function of

the execution hierarchy (500) of the activity (406), said notification (720) indicating that the identified one or more work items (422) being in the faulting state (608).

18. The computer-readable media (900) of claim 17, further comprising a transition component (916) for transitioning the remaining transitioned work items from the executing state (604) to the faulting state (608) in response to the transmitted notification (720).

19. The computer-readable media (900) of claim 15, further comprising a compensation component (918) for recovering data associated with the activity (406) as a function of the asynchronously handling the faulting event (608).

20. The computer-readable media (900) of claim 15, further comprising an inhibition component (920) for suppressing the transmission of the notification (720) to the remaining transitioned work items.

FIG. 1

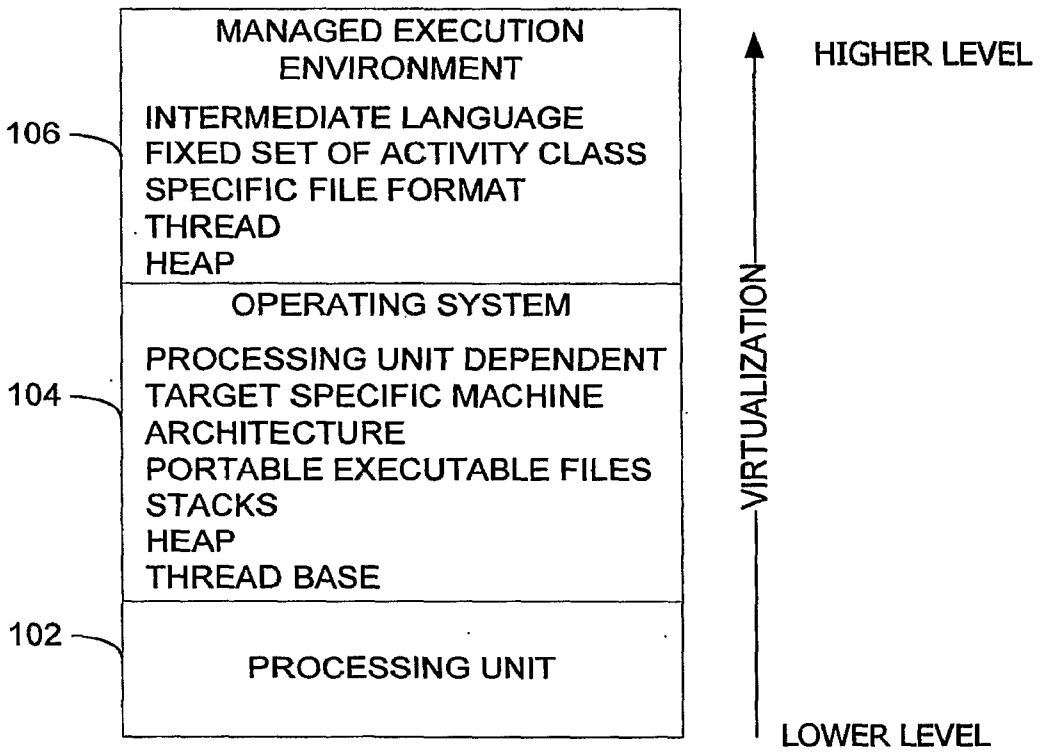

MANAGED EXECUTION ENVIRONMENT
INTERMEDIATE LANGUAGE
FIXED SET OF ACTIVITY CLASS
SPECIFIC FILE FORMAT
THREAD
HEAP
106
OPERATING SYSTEM
PROCESSING UNIT DEPENDENT
TARGET SPECIFIC MACHINE ARCHITECTURE
PORTABLE EXECUTABLE FILES
STACKS
HEAP
THREAD BASE
104
PROCESSING UNIT
102
HIGHER LEVEL
VIRTUALIZATION
LOWER LEVEL

FIG. 2

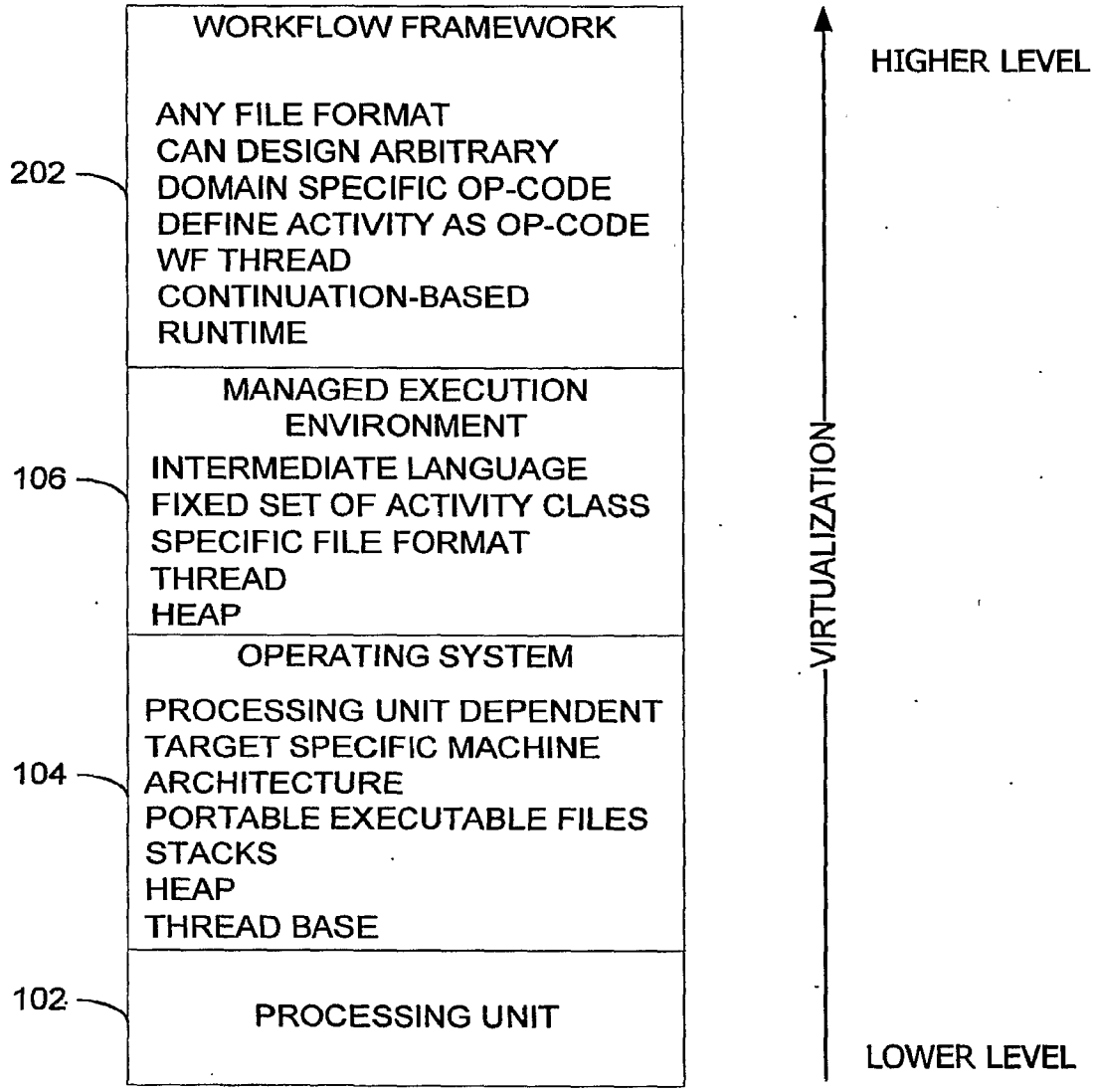

WORKFLOW FRAMEWORK
ANY FILE FORMAT
CAN DESIGN ARBITRARY
DOMAIN SPECIFIC OP-CODE
DEFINE ACTIVITY AS OP-CODE
WF THREAD
CONTINUATION-BASED
RUNTIME
202
MANAGED EXECUTION ENVIRONMENT
INTERMEDIATE LANGUAGE
FIXED SET OF ACTIVITY CLASS
SPECIFIC FILE FORMAT
THREAD
HEAP
106
OPERATING SYSTEM
PROCESSING UNIT DEPENDENT
TARGET SPECIFIC MACHINE
ARCHITECTURE
PORTABLE EXECUTABLE FILES
STACKS
HEAP
THREAD BASE
104
PROCESSING UNIT
102
HIGHER LEVEL
VIRTUALIZATION
LOWER LEVEL

FIG. 3

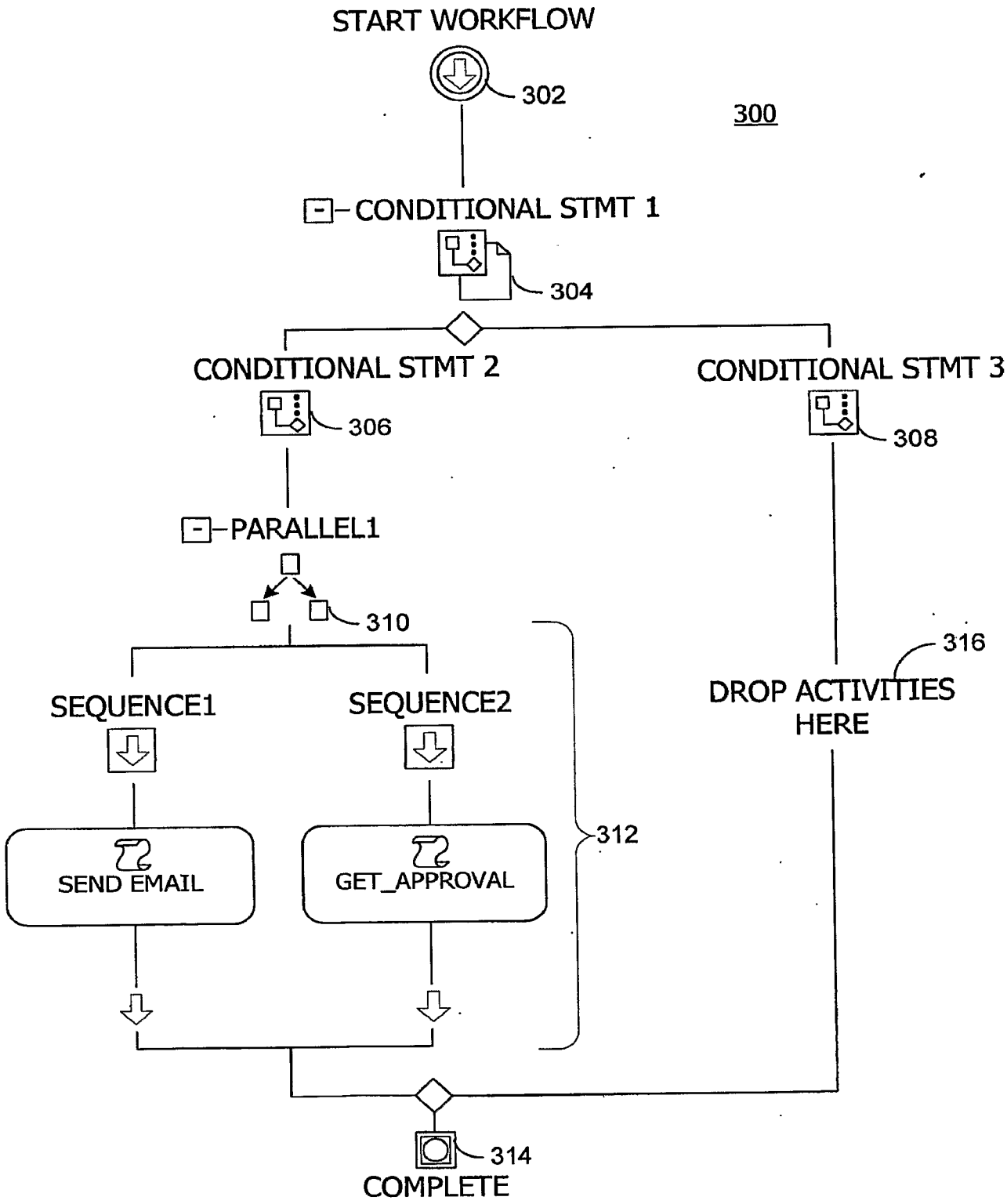

START WORKFLOW
302
300
CONDITIONAL STMT 1
304
CONDITIONAL STMT 2
306
CONDITIONAL STMT 3
308
PARALLEL1
310
316
SEQUENCE1
SEQUENCE2
DROP ACTIVITIES HERE
SEND EMAIL
GET_APPROVAL
312
314
COMPLETE

FIG. 4

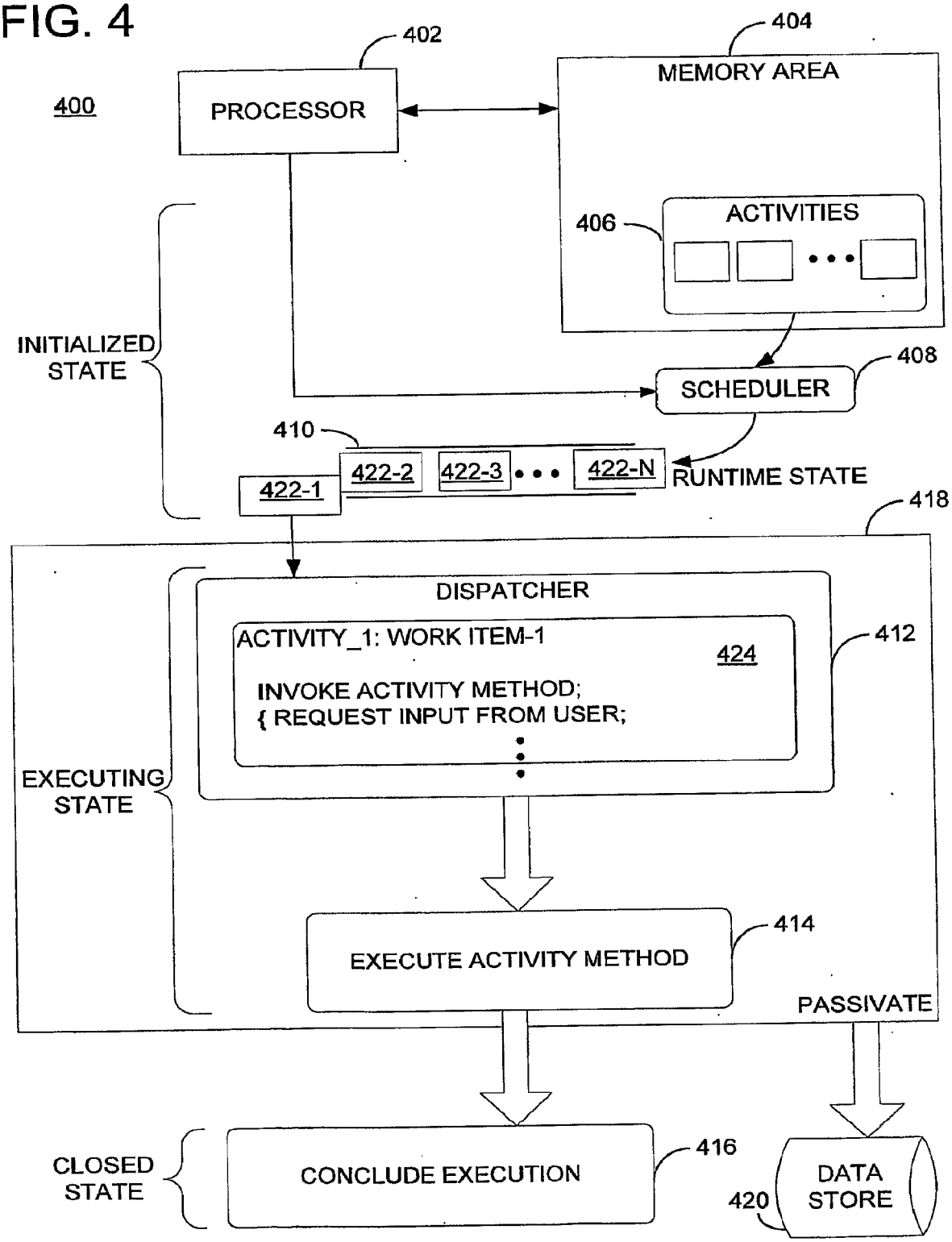
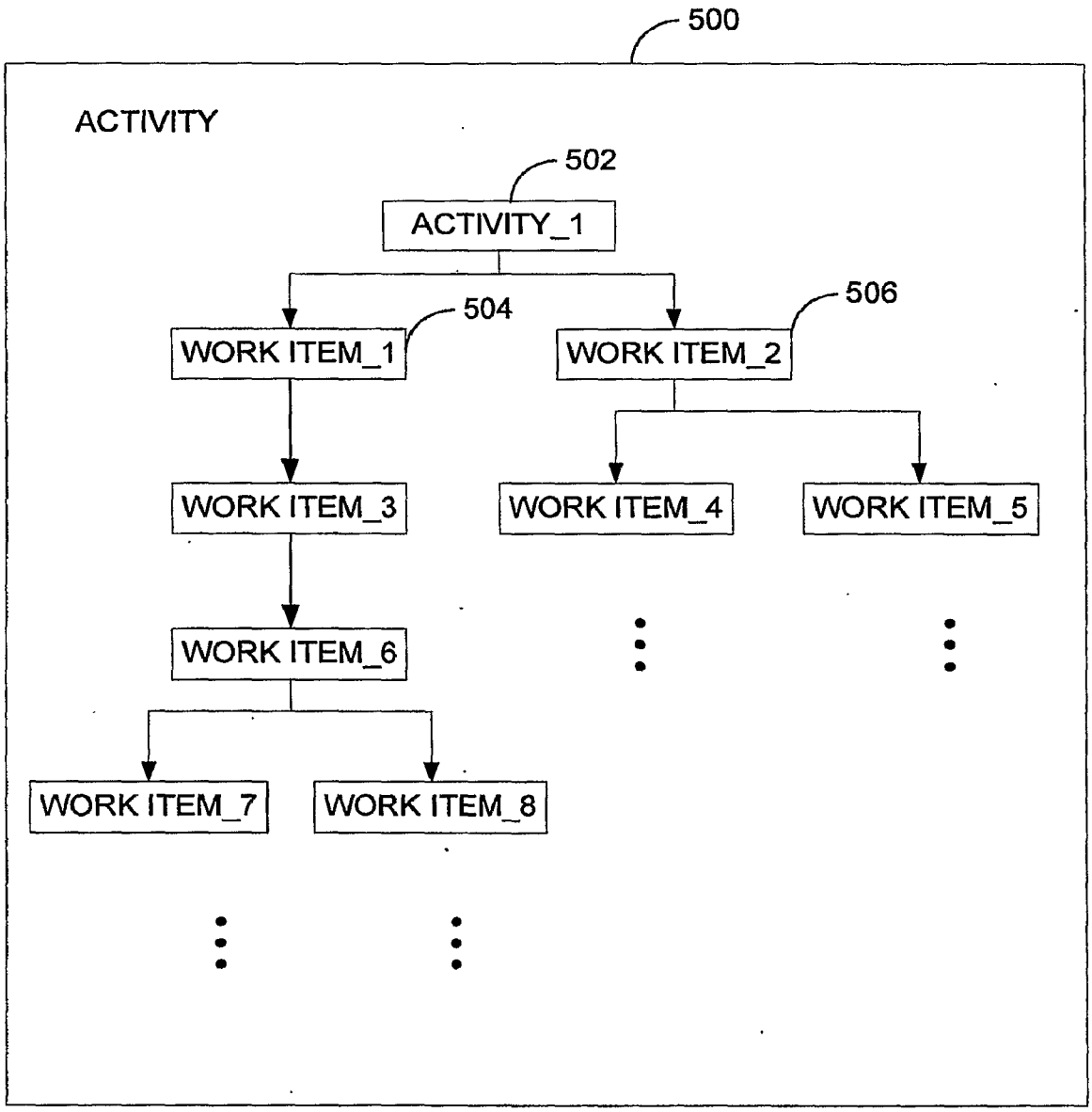

400
PROCESSOR
402
MEMORY AREA
404
ACTIVITIES
406
INITIALIZED STATE
SCHEDULER
408
410
422-1
422-2
422-3
422-N
RUNTIME STATE
418
DISPATCHER
ACTIVITY_1: WORK ITEM-1
424
412
INVOKE ACTIVITY METHOD;
{ REQUEST INPUT FROM USER;
EXECUTING STATE
EXECUTE ACTIVITY METHOD
414
PASSIVATE
CLOSED STATE
CONCLUDE EXECUTION
416
DATA STORE
420

FIG. 5


500
ACTIVITY
502
ACTIVITY_1
504
506
WORK ITEM_1
WORK ITEM_2
WORK ITEM_3
WORK ITEM_4
WORK ITEM_5
WORK ITEM_6
WORK ITEM_7
WORK ITEM_8

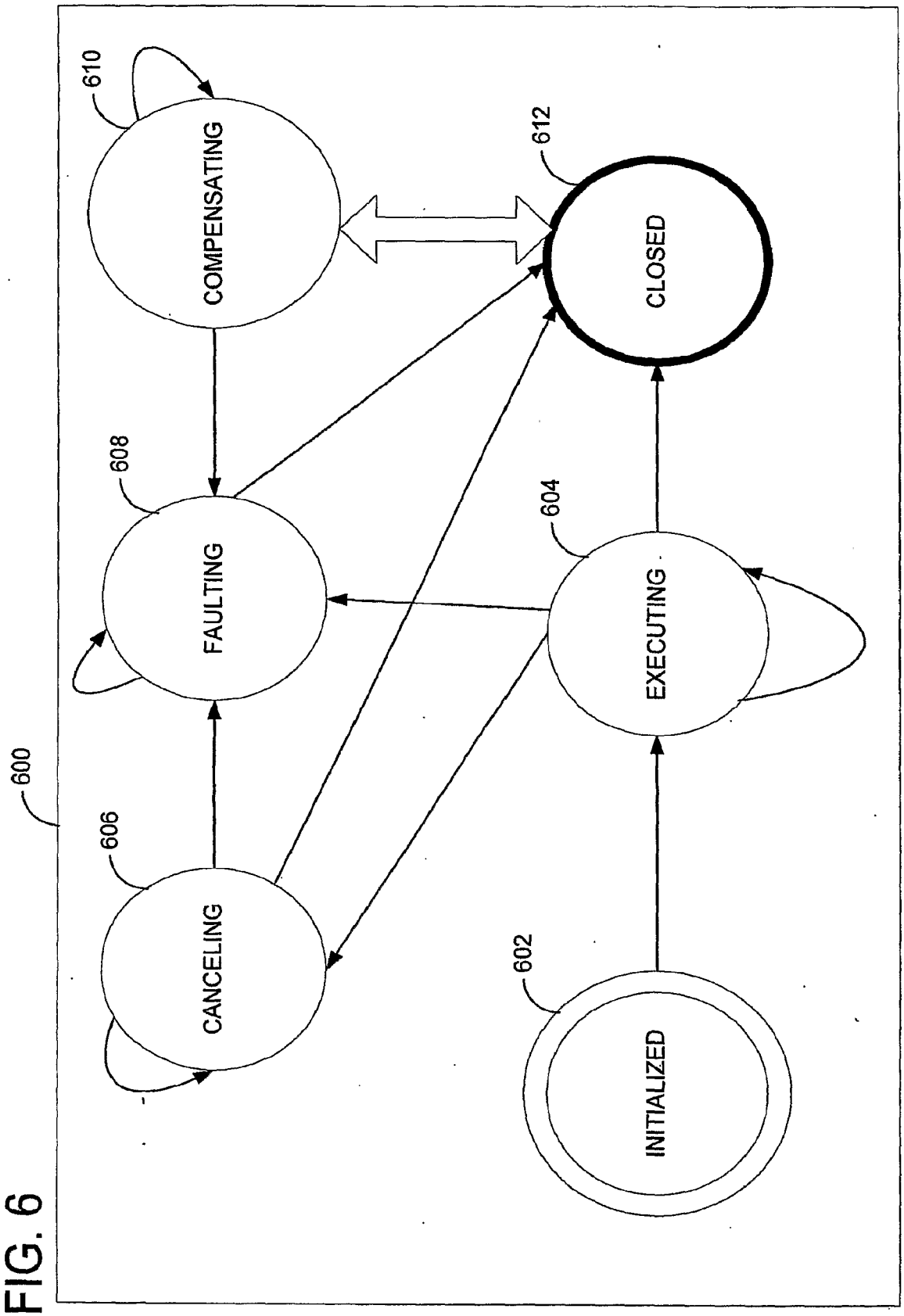

600
610
COMPENSATING
612
CLOSED
608
FAULTING
604
EXECUTING
606
CANCELING
602
INITIALIZED

FIG. 6

FIG. 7A

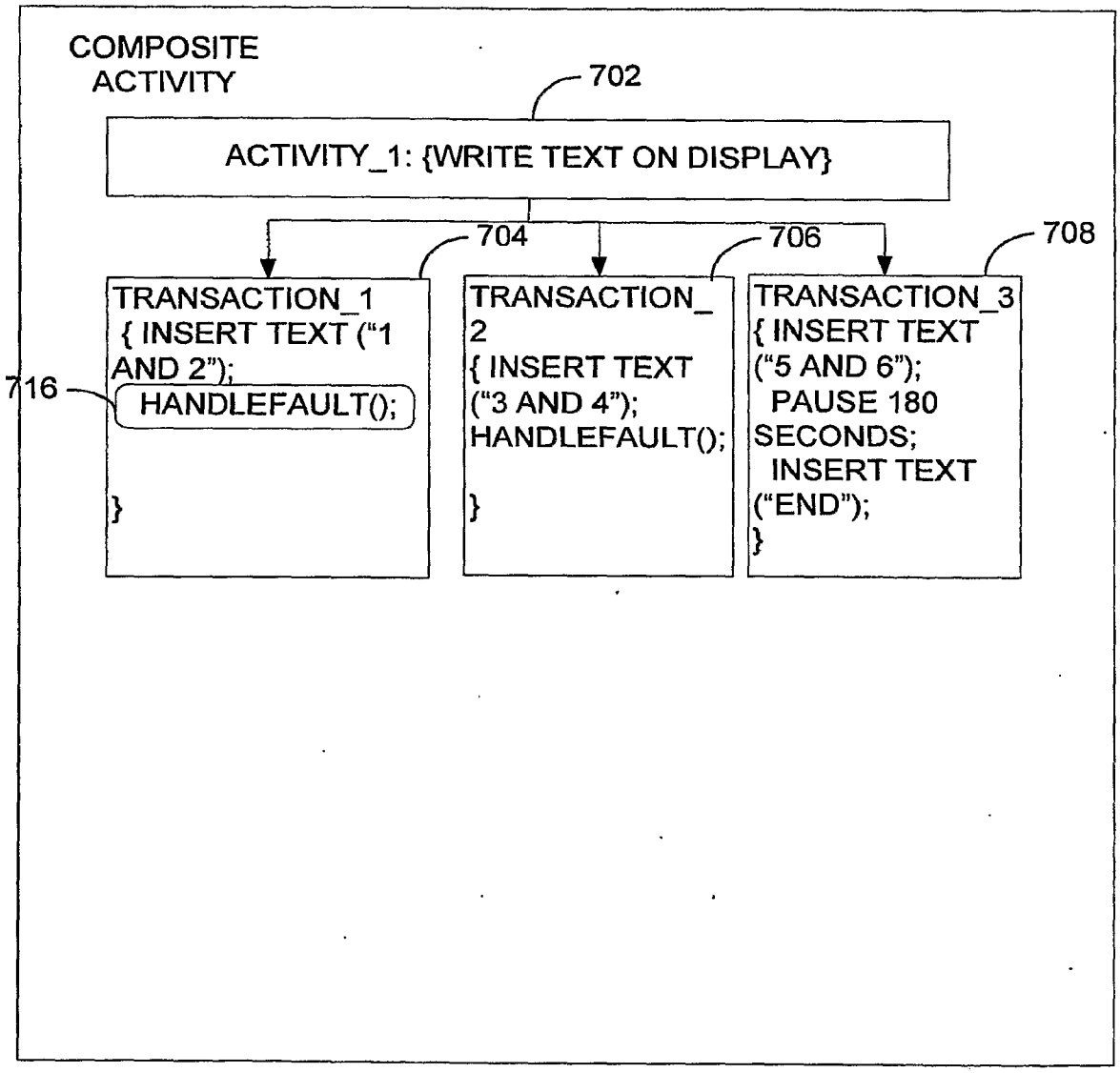

COMPOSITE ACTIVITY
702
ACTIVITY_1: {WRITE TEXT ON DISPLAY}
704
706
708
TRANSACTION_1 { INSERT TEXT ("1 AND 2");
716
HANDLEFAULT();
}
TRANSACTION_ 2 { INSERT TEXT ("3 AND 4"); HANDLEFAULT();
}
TRANSACTION_3 { INSERT TEXT ("5 AND 6"); PAUSE 180 SECONDS; INSERT TEXT ("END");
}

FIG. 7B

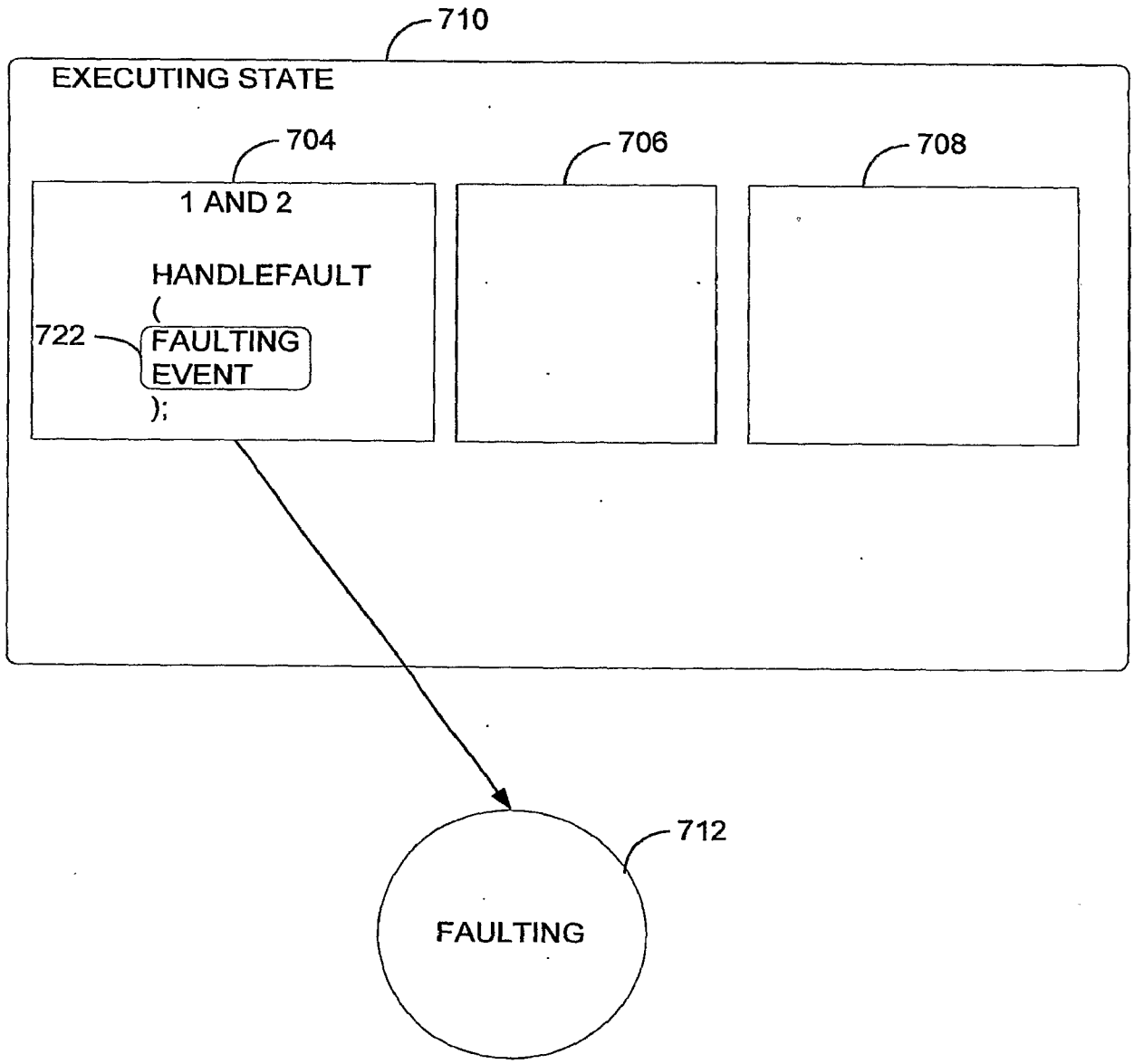

710
EXECUTING STATE
704
706
708
1 AND 2
HANDLEFAULT
(
722
FAULTING EVENT
);
712
FAULTING

FIG. 7C

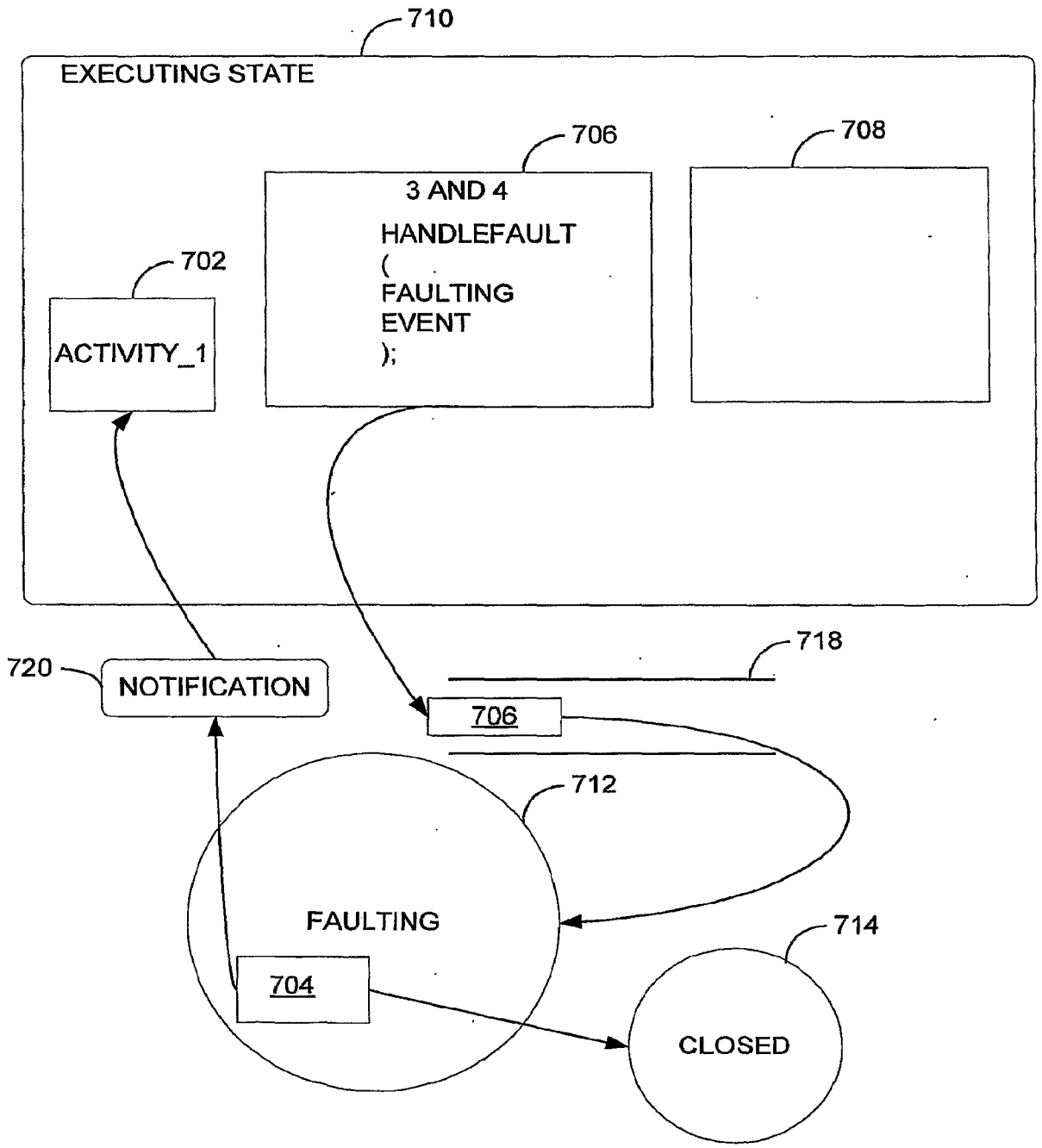

710
EXECUTING STATE
706
3 AND 4
HANDLEFAULT
(
FAULTING
EVENT
);
708
702
ACTIVITY_1
720
NOTIFICATION
718
706
712
FAULTING
704
714
CLOSED

FIG. 7D

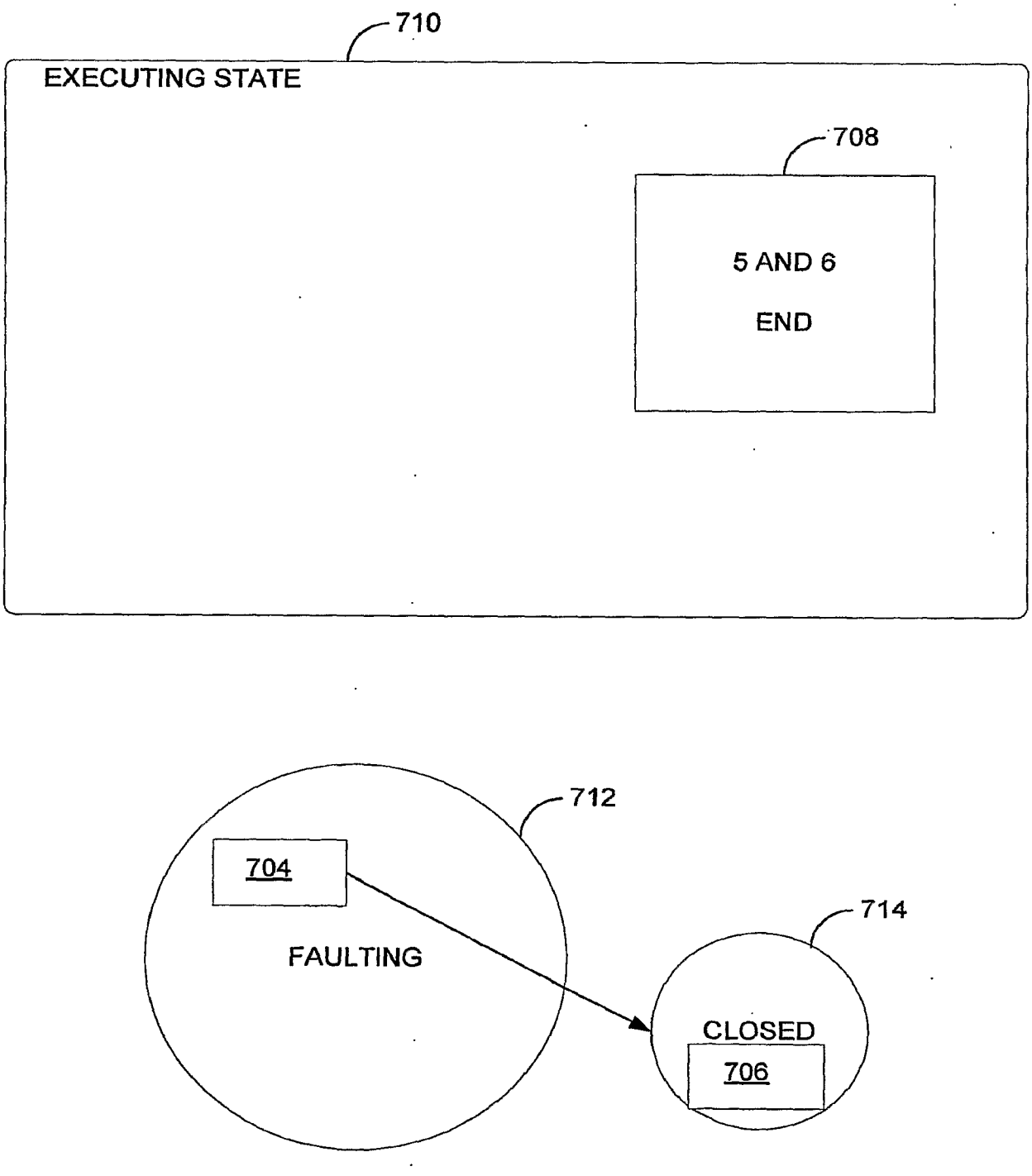

710
EXECUTING STATE
708
5 AND 6
END
712
704
FAULTING
714
CLOSED
706

FIG. 7E

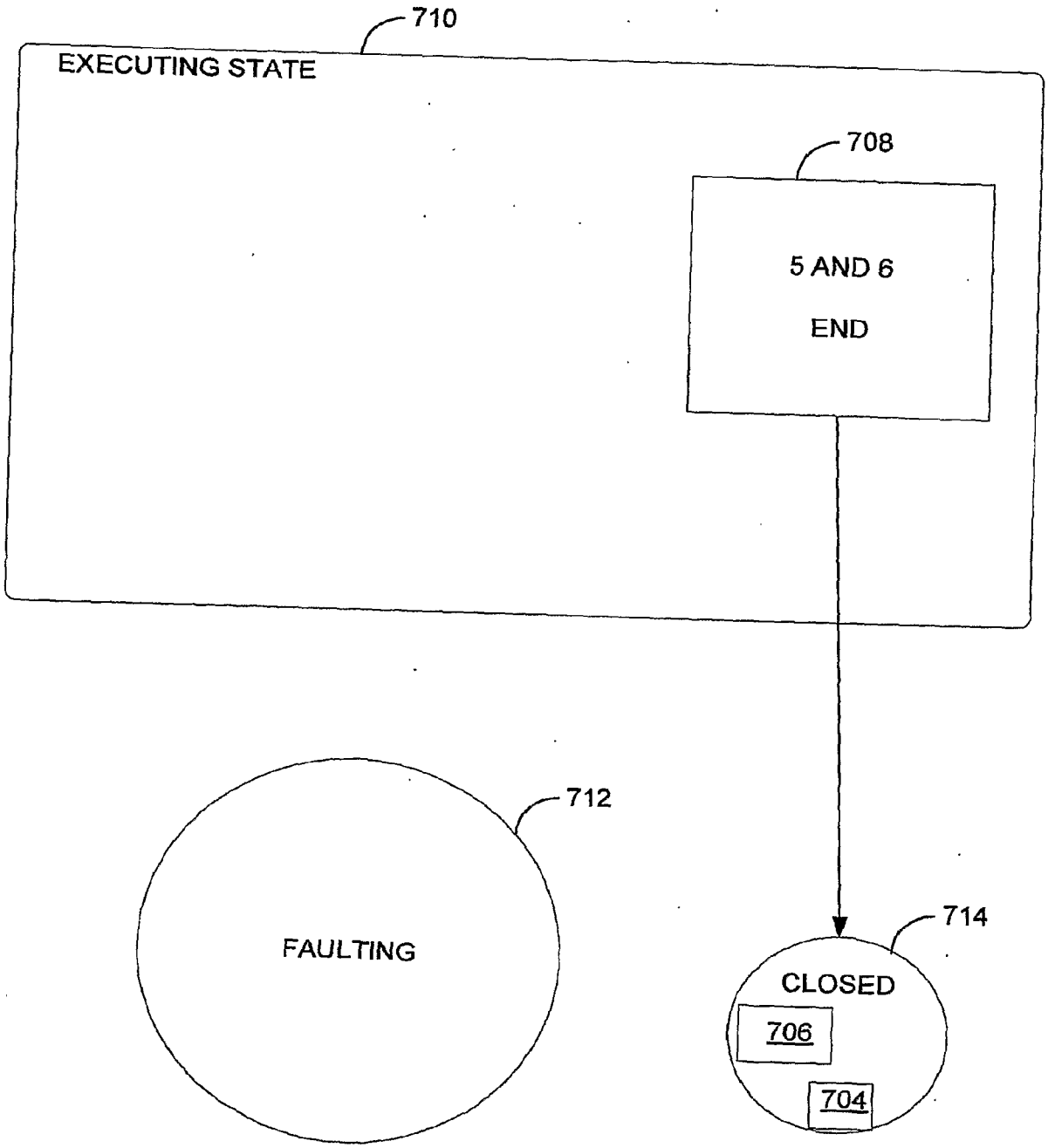

710
EXECUTING STATE
708
5 AND 6
END
712
FAULTING
714
CLOSED
706
704

FIG. 8

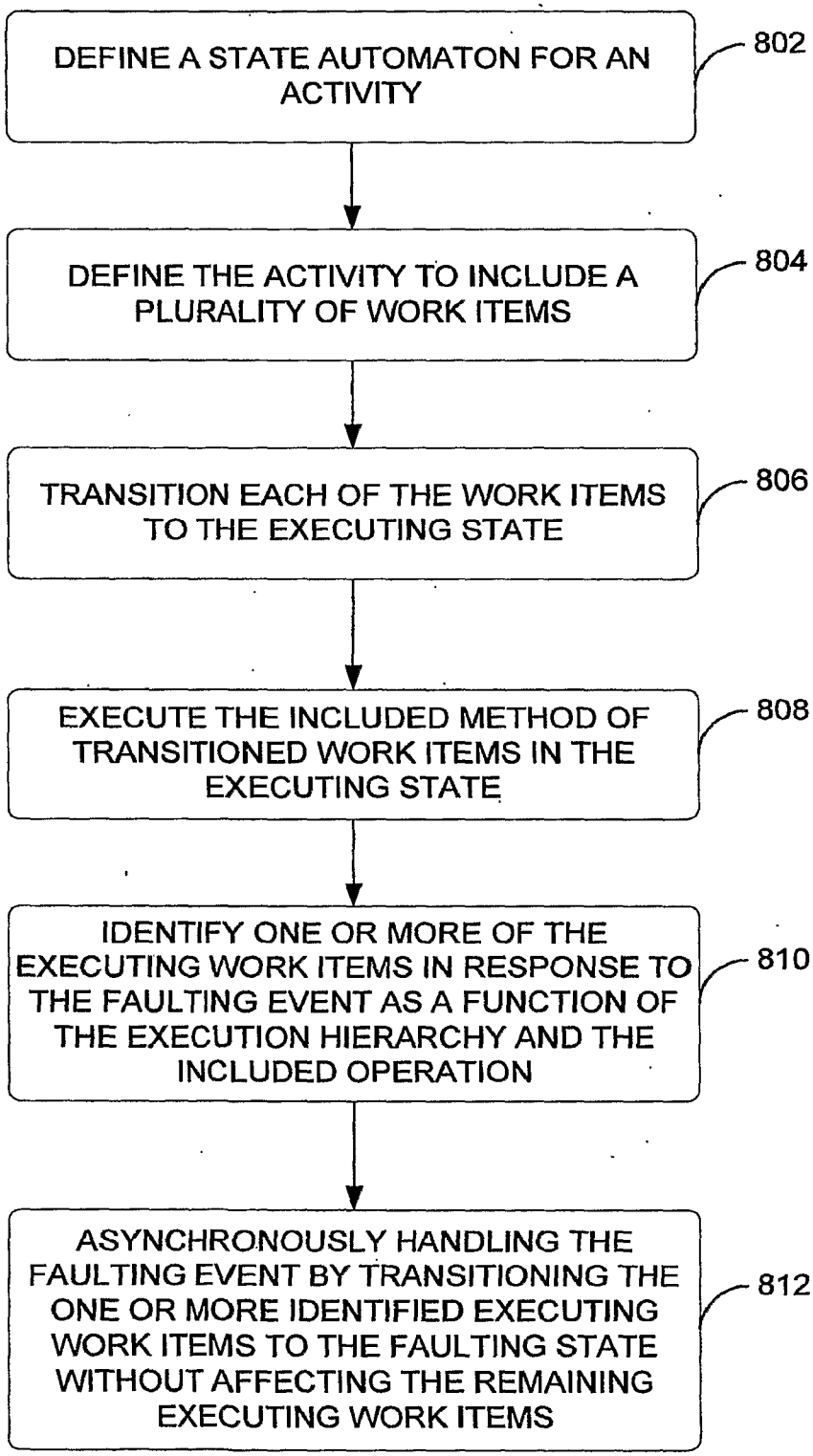
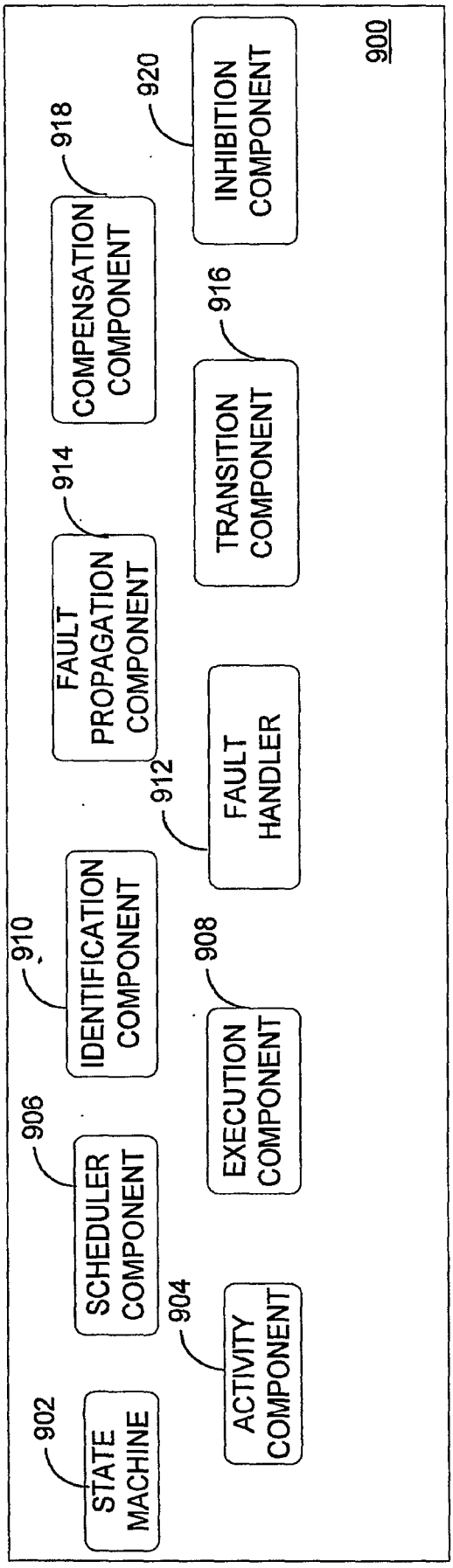

DEFINE A STATE AUTOMATON FOR AN ACTIVITY
802
DEFINE THE ACTIVITY TO INCLUDE A PLURALITY OF WORK ITEMS
804
TRANSITION EACH OF THE WORK ITEMS TO THE EXECUTING STATE
806
EXECUTE THE INCLUDED METHOD OF TRANSITIONED WORK ITEMS IN THE EXECUTING STATE
808
IDENTIFY ONE OR MORE OF THE EXECUTING WORK ITEMS IN RESPONSE TO THE FAULTING EVENT AS A FUNCTION OF THE EXECUTION HIERARCHY AND THE INCLUDED OPERATION
810
ASYNCHRONOUSLY HANDLING THE FAULTING EVENT BY TRANSITIONING THE ONE OR MORE IDENTIFIED EXECUTING WORK ITEMS TO THE FAULTING STATE WITHOUT AFFECTING THE REMAINING EXECUTING WORK ITEMS
812

FIG. 9


900
STATE MACHINE
902
ACTIVITY COMPONENT
904
SCHEDULER COMPONENT
906
EXECUTION COMPONENT
908
IDENTIFICATION COMPONENT
910
FAULT HANDLER
912
FAULT PROPAGATION COMPONENT
914
TRANSITION COMPONENT
916
COMPENSATION COMPONENT
918
INHIBITION COMPONENT
920

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2007/004642

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/44(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC08: G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models since 1975

Japanese utility models and applications for utility models since 1975

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

eKIPASS(Kipo Internal), Google, YesKisti

keyword: workflow , fault, faulty, fail, failure, exception, state activity, event, asynchronous, parallel

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	HAGEN, C. et al. Exception Handling in Workflow Management Systems. IEEE Transactions On Software Engineering. October 2000, Vol. 26, No.10, pages 943-958. See sections 5~7.	1-20
A	US 2004/0153350 A1 (KIM, Y.H. et al.) 05 AUGUST 2004 See figures 4,5,7 and their descriptions.	1-20
A	EP 0697652 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 21 FEBULARY 1996 See claim 1.	1-20
A	BRAMBILLA, M. et al. 'Exception Handling within Workflow-based Web Applications.' In: Web Engineering, 4th International Conference. Munich:LNCS Springer, 2004. (http://www.webml.org/webml/upload/ent5/1/213_brambilla_icwe2004.pdf) See sections 4.1~4.3.	1-20
A	US 2005/0193286 A1 (THATTE, S.R. et al.) 01 SEPTEMBER 2005 See figure 3 and its description.	1-20

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

☒ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
03 AUGUST 2007 (03.08.2007)	**03 AUGUST 2007 (03.08.2007)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office 920 Dunsan-dong, Seo-gu, Daejeon 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer YOON, Hye Sook Telephone No. 82-42-481-8370

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2007/004642

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2004/0153350 A1	05.08.2004	WO 2004/070527 A2	19.08.2004
EP 00697652 A1	21.02.1996	JP 08161161 A2 US 05819022 A	21.06.1996 06.10.1998
US 2005/0193286 A1	01.09.2005	NONE	