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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 February 2006 (02.02.2006)

PCT

(10) International Publication Number
WO 2006/010683 A1

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

(21) International Application Number:
PCT/EP2005/052998

(22) International Filing Date: 27 June 2005 (27.06.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/899,783 27 July 2004 (27.07.2004) US

(71) Applicant *(for all designated States except US)*: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York 10504 (US).

(71) Applicant *(for MG only)*: **IBM UNITED KINGDOM LIMITED** [GB/GB]; PO Box 41 North Harbour, Portsmouth Hampshire PO6 3AU (GB).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BOTZER, David** [IL/IL]; 26/30 Got Levin Street, 32922 Haifa (IL). **SHCHERBINA, Vladimir** [IL/IL]; 6/2 Lilah Street, 36821 Nesher (IL). **SHULMAN, Boris** [IL/IL]; 5 Paizer Street, 32896 Haifa (IL).

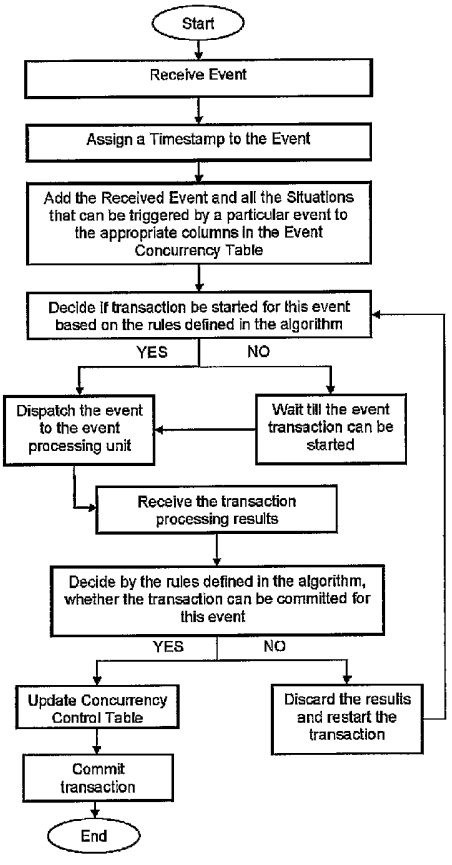
(74) Agent: **WILLIAMS, Julian, David**; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester Hampshire SO21 2JN (GB).

(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, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, 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,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR SCHEDULING PARTIAL ORDERED TRANSACTIONS FOR EVENT CORRELATION



(57) Abstract: A method and system for scheduling event transactions each having a respective time stamp, for use in an event-driven application. Possible time dependencies among events are resolved according to predeterOmined event rules associated with the events so as to allow determination as to when the event transactions may be processed. This allows the event transactions to be dispatched to distributed processors for concurrent processing.

WO 2006/010683 A1

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, 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).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

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.

Description

Method and system for scheduling partial ordered transactions for event correlation

Technical Field

This invention relates to event-driven systems and, in particular, to a method for ensuring the parallel processing of events, which are partially time-dependent, in the distributed environment.

Background Art

Reactive applications relate to a class of applica□tions that are event-driven and configured to operate upon detection of events. The exact timing and content of such events are not usually known in advance. Many tools in different areas have been built to detect events, and to couple their detection with appropriate actions. These tools exist in products that implement active databases, event management systems, the "publish/subscribe" mechanism, real-time systems and similar products.

US 6,604,093 (Etzion *et al.*) issued August 5, 2003 and entitled "*Situation awareness system*" is incorporated herein by reference and discloses a configurable event driven rules-based system for processing events in sequential order. Throughout the remainder of this document, this system will be referred to by its acronym "AMIT" meaning "Active Middleware Technology." AMIT is a trademark of International Business Machines Inc. of Armonk, New York, USA. That is, incoming events are processed one by one, while all relations (both temporal and quantitative) among different events' instances are kept in memory. While this approach is absolutely necessary for the case, where most of the events have temporal relations (e.g. a specific event's sequence occurs within some specified period of time), it will be not scalable for other cases, where a huge number of temporally independent events should be processed in the distributed environment (e.g. aggregation event rules, where the order of single event instances are less important then their calculated statistical metrics). Also, if AMIT event rules (event combinations) include database queries, then the ACID transaction support should be added.

US 6,272,515 (Fouquet) issued August 7, 2001 and entitled "*Method of scheduling distributed transactions*" is incorporated herein by reference and discloses a method of scheduling distributed transactions for a transactional monitor with the possibility of parallel initiation of transactions in order to serialize activation of the operations of the transactions in accordance with the chronological order in which the transactions are initiated and allowing for conflicts of execution among operations of different transactions or of the same transaction. Thus, while such method is applicable to con-

ventional transactions that can be divided into elementary operations and events signal only a transaction initiation or an operation termination, such an approach is not suitable for those rule engines that detect pre-defined events combinations (event driven rules) for subsequent processing in distributed transaction environments.

In such case, an event driven rule engine handles every event instance as an atomic transaction that cannot be further divided to composite operations. Such a transaction should lock a resource that is specific for the handled event instance. The resource is a data structure that represents a computational window, which is a time period, where the event instance can be accounted and where a predefined event's combination (rule) can occur. In case of temporal and other semantic dependencies among incoming events, there are possible conflicts among different distributed transactions that should be resolved.

Disclosure of Invention

It is therefore an object of the invention to provide an improved method and system for the parallel processing in a distributed, transaction-oriented environment of event transactions having event-driven rules defining events that are semantically related, where the order of the constituent events can be important.

In accordance with a broad aspect of the invention, there is provided a method for scheduling event transactions each having a respective time stamp, for use in an event-driven application having predetermined event rules, said method comprising: resolving possible time dependencies among events according to said predeter□mined event rules; and determining when the event transactions may be processed thus allowing the event transactions to be dispatched to distributed processors for concurrent processing.

Viewing the present invention from another aspect, there is now provided a system for implementing such a method. Preferably, such a system comprises an event receiver and conveyor that detects pre-defined event combinations and comprises: an event transaction for receiving one or more events; an event concurrency control unit coupled to the event transaction and responsive to respective time stamps associated with said events for resolving possible time dependencies among events according to associated event rules; and at least two output ports of the event concurrency control unit being adapted to be coupled to respective distributed processors for executing said events in parallel, according to dependencies that are learned from said event rule definitions.

Thus, preferred embodiments of the present invention defines transactions that suit the event rule giving rise to a situation in event driven rules engines, and defines a concurrency control mechanism for such transactions.

Processing the transactions concurrently rather than sequentially requires the transactions to be distributed between several event processing units.

The event concurrency control unit centrally executes a concurrency control algorithm using a single machine and dispatches transactions that are ready to run, i.e. all transactions that were supposed to be executed before the current transaction finished executing, to different processing units. The processing units can be different processes on different machines or different processes (threads) on a single machine using remote procedure calls (RPC). The invention was developed for AMIT, but can be easily applied to the whole class of event driven rule engines operating in distributed environments.

Brief Description of the Drawings

In order to understand the invention and to see how it may be carried out in practice, preferred embodiments thereof will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

Fig. 1 is a block diagram showing schematically a system according to a first embodiment of the invention for detecting and processing events according to pre-defined event combinations (rules);

Fig. 2 is a block diagram showing schematically a system according to a second embodiment of the invention for use in a distributed environment;

Figs. 3, 4 and 5 show dependency graphs describing dependencies between events and situations according to different examples according to the invention;

Fig. 6 is a flow diagram showing the principal actions carried out by an event receiver and conveyor according to the invention; and

Fig. 7 is a block diagram showing functionally in greater detail the event receiver and conveyor depicted in Fig. 1.

Best Mode for Carrying Out the Invention

Fig. 1 is a block diagram showing schematically a system 10 according to a first embodiment of the invention for detecting and processing events according to pre-defined event combinations (rules). An event dispatcher 11 conveys events sequentially to an event concurrency control unit 12 according to a timestamp associated with each event, and an event processing unit 13 starts event processing for each event. The event processing unit 13 can be any conventional event transaction processor such as AMIT described in above-referenced US 6,604,093. The event concurrency control unit 12 employs a timestamp based concurrency control algorithm (described below) that uses the semantic knowledge of event rules in order to process concurrently incoming events. It checks if event transaction can be started according to the current event rules and suspends the transaction's execution if necessary. It also commits or

aborts already running event transactions in the case that their corresponding events were defined for complex (nested) event combinations, which were derived from other event combinations and elementary events.

The event rules may be stored in an external database 14 or in a data file that is accessed by the event dispatcher 11 in an analogous manner to the way in which events and rules are conventionally fed to AMIT. However, whereas the events are conventionally fed to AMIT directly, in the invention they are fed not directly but rather via the event dispatcher 11. The rules may be read by AMIT and the event dispatcher simultaneously. The event dispatcher 11 first feeds the events to the event concurrency control unit 12 and only then feeds them to the event processing unit 13 in response to initiation control information received from the event concurrency control unit 12. The time□stamp information associated with each event may be derived by an internal clock (not shown) or by a pre-defined running order associated with each event. The event dispatcher 11 in conjunction with the event concurrency control unit 12 operates as an event receiver and conveyor 15 that processes incoming events so as to determine when their processing may be started according to known dependencies and then dispatches them to an event processing unit 13.

Fig. 1 shows the construction of the event concurrency control unit 12 and the manner in which it is coupled to the event processing unit 13. In practice, a preferred embodiment of the invention can run several event processing units concurrently, so in practice the event concurrency control unit 12 has multiple processor output ports each adapted for coupling a respective event processing unit thereto.

Fig. 2 is a block diagram showing schematically a system 20 according to such an embodiment of the invention for use in a distributed environment. Components that have already been described with reference to Fig. 1 will be identified by the same reference numerals. Thus, in the system 20 the event receiver and conveyor 15 is coupled to multiple event processing units, each of which is designated 13 and may be realized by the AMIT engine. Incoming events are processed by the event receiver and conveyor 15 as described above with reference to Fig. 1 so as to determine when they may be initiated. They may then be dispatched to different event processing units 13 for concurrent processing although the manner in which this is done and the manner in which different processors are selected are not themselves features of the invention.

Having described possible implementations of a system according to the invention, the manner in which the timestamp based concurrency control algorithm operates will now be described.

A dependency graph among all events and situations in the system is created. The dependency graph may be constructed based on analysis of the metadata (rules definitions) that are normally fed as direct input to the event processing units 13 but in

the invention are fed thereto indirectly via the event receiver and conveyor 15. For the purpose of implementing an embodiment of the invention, it is assumed that as part of the rules definitions the following data structures are maintained for each event:

List of situations to which this event should be added;

List of situations for which this event can start their computation window;

List of situations for which this event can close their computation window;

List of situations that can be triggered by this event.

An algorithm embodying the invention processes these rules to determine the dependencies between events and situations and between situa□tions and other situations in order to construct data representative of the event/□situation hierarchy that includes data indicative of timestamps associated with constituent events. This information is then used to determine whether an event can legitimately have started or is dependent on the completion of a related event before it can start. Within this context, it will be understood that a situation is merely one kind of event that is caused by an event or a combination of events. Therefore, no distinction is made between the term "event" and "situation" so far as the algorithm is concerned.

The algorithm allows optimistic processing of events whereby events are processed concurrently by distributed processors, rather than sequentially as is done conventionally, so as to allow for the possibility that a later event can commence processing without the need to wait for earlier events to commit so as not to delay processing of a later event that is not time-dependent on the earlier event. On the other hand, if it is determined from the event hierarchy that the later event cannot be resolved pending the completion of an earlier event, then the later event will be aborted until the earlier event on whose completion it is time-dependent has terminated.

The algorithm processes the event data as will be explained below so as to extract the time dependencies between events. In order to explain how this is done in a manner that will be more intuitive, the event hierarchy will be represented graphically. But it is to be understood that, in practice, what is important is the information implicit in the graphical representation and since the information is anyway inherent in the event data, the algorithm need not actually produce any graphical representation of the event hierarchy. Thus, the following rules are applied in order to construct the event hierarchy graphically:

If an event or situation participates in a situation as an operand, then an edge is added from the event/situation node to the situation node.

If an event or situation acts as the start or end of a computation window, then the corresponding start/end edge is added from the event/situation node to the situation.

A concurrency control table is created, wherein a separate column is defined

for each situation definition. If there is an edge from an event to the situation associated with a column of interest in the dependency graph, then the event is added to the column. Each column is sorted by an event's arrival timestamp. Events are added to appropriate columns in the concurrency control table with their arrival timestamp. All situations that can be triggered by a particular event are added to appropriate columns in the concurrency control table, with a mark that indicates that those situations are still pending. In the examples described below, the mark is denoted by an asterisk (*). For events that act as an end of a computation window, a special mark is added in order to take care of deferred/delayed situations. In the examples described below, the path from such events to a dependent situation is marked as "End Edge." In similar manner, the path from events that act as a start of a computation window to a dependent situation is marked as "Start Edge." This allows the algorithm to identify start and end dependencies of computation windows relating to dependent situations.

An event transaction should start, if it is placed first among "real" events (not marked situations) in all the columns of the concurrency control table. The event transaction will be committed only if it is placed first in all the columns of the concurrency control table (among all events and marked situations in the table). In the case that there is a regular event and an event that acts as an end of the computation window, with the same timestamps (in the case of deferred/delayed situations) the event will start the new transaction. When an event transaction with a timestamp t2 attempts to commit, the algorithm checks if any marked situation having a timestamp t1 (t1<t2) became real (the mark was removed). In such case, the later event transaction is aborted and re-started.

If there is a situation with such timestamp that is still marked, the transaction processing will be blocked till the marked situation is resolved (removed or changed to real).

Different examples will now be described with reference to Figs. 3 to 5 of the drawings. Fig. 3 refers to the simple case of dependencies, where the first event leads to the detection of a situation, and the detected situation participates in other situation detection together with a second event. Fig. 4 illustrates the case, where incoming events can also close the computational window for detecting situations and Fig. 5 refers to the case, where detected situations reported only at the end of computational window period, but still should be taken into the account for other situations detections.

Example 1:

Fig. 3 shows graphically a situation S1 that is a composite function of e1, and a situation S2 that is a composite of S1 and e2. It is assumed that the event transactions are partially ordered, implying that the order or sequence in which events are committed is important. The computation windows for both situations are started at startup and never ended.

The scenario:

Event e1 arrives at t1, event e2 arrives at t2 (t2>t1). The concurrency control table for S1 and S2 will be:

Table 1

S1	S2
e1;t1	s1;t1*
	e2;t2

So far as situation S2 is concerned, S1 is a constituent event that must complete (i.e. commit) before situation S2 can occur. However, S1 is also dependent on the event e1 committing. This means that so far as situation S2 is concerned, the event s1 is unresolved at time t1 and so is marked. Both e1 and e2 will start transactions concurrently. If s1 is triggered, e2 will abort its transaction and will redo its work. This ensures that ordering of transactions is maintained if S2 is a sequence of (S1, e2) indicating that S1 must commit before e2 commits and that only then can S2 be resolved. On the other hand, there may be scenarios where S1 does not occur and is deleted from the concurrency control table. For example, S1 might be the situation "bank account is overdrawn" and e1 being the event spawning this situation might simply be a customer's current balance. Thus, if by the time e2 has committed, e1 arrives and is a healthy balance, S1 will not be triggered and can be deleted from the concurrency control table. This can happen for definitions that require only one event in order to trigger a situation. For instance S2 may be defined to be triggered if one of two events arrives (S1 or e2). Consider, for example, that S2 denotes the situation "account problem" and e2 indicates that "credit card is cancelled." So in this case if S1 is not triggered then e2 will not be rolled back and S2 will be detected. This allows the event dispatcher 11 to dispatch the event transactions e1 and e2 to different processors so as to allow parallel processing thereby saving time.

Example 2:

Fig. 4 shows graphically a situation S1 that is a composite function of e1. Situation S2 is a composite function of S1 and e2. Thus far, the event hierarchy is the same as that described above with reference to Fig. 3. However, while all situations occur in specific computation windows, in previous cases the computation windows were

unlimited and extended from startup without termination, the situations shown in Fig. 4 are constrained to occur within computation windows that commence after startup and have predetermined terminators. These are shown graphically as start or end edges associated with an event that indicate that a situation spawned by that event is constrained to occur with a computation window that either starts or ends concurrently with the event, depending on whether the edge is a start edge or an end edge, respectively. Thus, event e3 ends a computation window for both S1 and S2 meaning that when e3 occurs, computation of both S1 and S2 ends. On the other hand, the computation windows of S1 and S2 are both started on arrival of e4. Thus, the computation windows for both S1 and S2 extend from the time associated with e4 to the time associated with e3. This means that if e3 arrives before e4, it has no influence on S1 because its computation window is not started yet since S1 would need to be computed before its computation starts. Event e4 still determines that computation of S1 can commence and it completes when the computational window is closed by e3. Event e4 starts the computation window, but S1 will be detected as soon as suitable after arrival of event e1 (commensurate with the threshold conditions if there are any). Thus, when e4 arrives it will start the computation window of S1, and the next e1 event that arrives (but before e3) will be computed. By way of example, e4 may be a "day start" event and e3 a "day end" event. S1 may be "bank account is overdrawn", event e1 indicating the "account balance". Event e2 may be "Request for payment" and S2 could be "Payment Failed", so that "Payment Failed" will be detected if in the same business day the "Request for payment" and "account overdrawn" are received.

The scenario:

Event e4 arrives at t1, event e1 at t2, event e3 at t3 and event e2 at t4, where t1<t2< t3<t4.

In this case the concurrency control table will contain:

Table 2

S1	S2
e4start;t1	e4start;t1
e1;t2	s1;t2*
e3end;t3	e3end;t3
	e2;t4

In this case the processing will be sequential, because e3 cannot start before e1 (S1), and e2 cannot start before e3 (S2).

Closing a computation window

If an event closes the computation window, the transaction will start only if it

placed first among all events (both real and those "marked situations" which are processed optimistically but may need later to be aborted and re-run), and will be committed for each situation separately (partial commit). After partial commitment, other transactions may start for the committed situation. Also after the closing computation window for a situation, the method checks if delayed/deferred situations were triggered and were written in the table before the event that closes the window (with same timestamp). This will occur in the case of deferred/delayed situations that are needed in other situations whose computational windows are closed by the same event.

E xample 3:

Fig. 5 shows graphically a deferred situation S1 that is a composite function of e1. Situation S2 is a composite function of S1. Thus here also, the event hierarchy is the same as that described above with reference to Fig. 3. However, computation windows for both situations are started at startup and ended on arrival of event e2.

The scenario:

Event e1 arrives at t1, e2 arrives at t2 (t1<t2).

In this case the concurrency control table will contain:

Table 3

S1	S2
e1;t1	s1*;t2
e2end;t2	e2end;t2

Situation s1 is not written with timestamp t1 because only e2 can trigger S1 (deferred situation). S1 must be committed within a computation window that ends with the arrival of e2. Thus, when e2 arrives, computation of S1 stops, i.e. its computation window closes. The computation of S1 is executed when e2 arrives, i.e. since S1 is a deferred situation, its computation is deferred until e2 arrives. The actual computation is done on e1 events that arrived before e2.

When e2 arrives and starts its process, the computation of S1 ends. If S1 is now detected, implying that the situation has occurred, then S1:t1 is written to the concurrency control table and e2 will suspend its processing, and will resume its processing only after S1's process is completed. This allows computation of deferred situations that are a part of AMIT language, for example.

An example of a deferred situation is a NOT situation. For example, suppose situation S1 is defined as NOT(e1). In this case S1 will be detected if e1 does not arrive before e2. Event e1 will be as before the "account balance" situation S1 will be "Average Balance" and will be defined as Report(e1). This situation will calculate the

maximal balance till event e2 is received, and it will be triggered on arrival of e2. Situation S2 will be "Good Day" and it will be triggered if the average balance calculated by S1 is greater than a certain amount.

Fig. 6 is a flow diagram summarizing the principal actions carried out by the event receiver and conveyor 15 shown in Figs. 1 and 2. Thus, on receiving an event, the event receiver and conveyor 15 assigns a timestamp to the event and adds the received event and all the situations that can be triggered thereby to the appropriate columns in the event concurrency table. It then decides in accordance with predetermined rules whether the transaction can be started for this event. If not, the event receiver and conveyor 15 waits until the event transaction can be started. If and when the event transaction can be started, the event receiver and conveyor 15 dispatches the event to the event processing unit 13 from which it then receives the transaction processing results. It then decides in accordance with the predetermined rules, whether the transaction can be committed for this event. If not, it discards the results and restarts the transaction. If so, it updates the event concurrency table and commits the results. It should be noted that the selection of a processor is not a part of the invention. Any suitable existing clustering mechanism can be used. For example in the case of a WebSphere Aplication Server based solution, then a WebSphere cluster may be used, and the processor will be chosen by its load balancing mechanism.

Fig. 7 is a block diagram showing functionally in greater detail an event receiver and conveyor 15 comprising an event dispatcher 11 for receiving one or more events. An event concurrency control unit 12 is coupled to the event dispatcher and is responsive to respective time stamps associated with the events for resolving possible time dependencies among events according to the associated event rules. The time stamps may either be prior associated with the respective events before they arrive or, alternatively, time stamps may be assigned to the events by the event receiver and conveyor 15 on arrival. The event dispatcher 11 includes a registration unit 20 for registering each incoming event so as to maintain a list of registered events and is adapted to dispatch each event transaction for which all dependencies are resolved to an event processing unit 15, thus allowing parallel execution of events, according to dependencies that are learned from event rule definitions associated with the events. The event concurrency control unit is configured to resolve possible time dependencies for each registered event with registered events and also with marked situations that are not detected yet according to the predeter□mined event rules.

The event receiver and conveyor 15 further includes a receiver unit 21 for receiving a processing result of the event transaction and a status determina□tion unit 22 coupled to the receiver unit 21 for determining whether the event transaction is to be committed or aborted. An interruption unit 23 coupled to the status determination unit is

responsive to a status that indicates that the event transaction is to be aborted, for aborting the event transaction. The event dispatcher 11 is coupled to the interruption unit for dispatching the event trans□action again to a processing unit. The processing unit may be the same as the processing unit or may be a different processing unit.

It will also be understood that the system according to the invention may be a suitably programmed computer. Likewise, the invention contemplates a computer program being readable by a computer for executing the method of the invention. The invention further contemplates a machine-readable memory tangibly embodying a program of instructions executable by the machine for executing the method of the invention.

Claims

A method for scheduling event transactions each having a respective time stamp, for use in an event-driven application having predetermined event rules, said method comprising: resolving possible time dependencies among events according to said predeter□mined event rules; and determining when the event transactions may be processed thus allowing the event transactions to be dispatched to distributed processors for concurrent processing.

A computer system comprising at least two processors for processing distributed event transactions each having a respective time stamp, in a parallel distributed manner, said system comprising a scheduler for scheduling event transactions using the method according to claim 1.

The method according to claim 1, wherein determining when the event transactions may be processed comprises: registering each incoming event so as to maintain a list of registered events; for each registered event resolving possible time dependencies among registered events and also with marked situations that are not detected yet according to said predeter□mined event rules; dispatching each event transaction for which all dependencies are resolved to an event processing unit; receiving a processing result of the event transaction; determining whether said event transaction is to be committed or aborted; and if the event transaction is to be aborted, aborting the event transaction and dispatching it again to a processing unit.

The method according to claim 3, wherein registering each incoming event includes: receiving an event; assigning a timestamp to the event; determining using predetermined rules all the situations that can be triggered by the received event; adding the received event and all the situations that can be triggered thereby to appropriate columns in an event concurrency table; determining in accordance with said predetermined rules whether a transaction spawned by the received event can be started for the received event; if the transaction cannot be started for the received event, waiting until the transaction can be started; if the transaction can be started: dispatching the event to an event processing unit; receiving transaction processing results from the event proces□sing unit; determining in accordance with the predetermined rules, whether the transaction can be committed for the received event; if the transaction cannot be committed for the received event, discarding the processing results and restarting the transaction; if the transaction can be committed for the received event, updating the event concurrency table and committing the transaction.

The method according to claim 3, wherein registering each incoming event

includes: receiving an event having a timestamp associated therewith; determining using predetermined rules all the situations that can be triggered by the received event; adding the received event and all the situations that can be triggered thereby to appropriate columns in an event concurrency table; determining in accordance with said predetermined rules whether a transaction spawned by the received event can be started for the received event; if the transaction cannot be started for the received event, waiting until the transaction can be started; if the transaction can be started: dispatching the event to an event processing unit; receiving transaction processing results from the event proces□sing unit; determining in accordance with the predetermined rules, whether the transaction can be committed for the received event; if the transaction cannot be committed for the received event, discarding the processing results and restarting the transaction; if the transaction can be committed for the received event, updating the event concurrency table and committing the transaction.

The method according to claim 3, wherein situations are constrained to occur within computation windows that commence after startup and have predetermined terminators, there being further included: associating with an event data that indicates that a situation spawned by that event is constrained to occur with a computation window that either starts or ends concurrently with the event.

An event receiver and conveyor comprising: an event dispatcher for receiving one or more events and being adapted to be coupled to respective distributed processors for executing said events in parallel, according to dependencies that are learned from event rule definitions associated with the events; and an event concurrency control unit coupled to the event dispatcher and responsive to respective time stamps associated with said events for resolving possible time dependencies among events according to the associated event rules.

The event receiver and conveyor according to claim 7, wherein: the event dispatcher includes a registration unit for registering each incoming event so as to maintain a list of registered events and is adapted to dispatch each event transaction for which all dependencies are resolved to an event processing unit; the event concurrency control unit is configured to resolve possible time dependencies for each registered event with registered events and also with marked situations that are not detected yet according to said predeter□mined event rules; the event receiver and conveyor further including: a receiver unit for receiving a processing result of the event transaction; a status determination unit coupled to the receiver unit for determining whether said event transaction is to be committed or aborted; and an interruption unit coupled to the status de-

termination unit and being responsive to a status that indicates that the event transaction is to be aborted, for aborting the event transaction; the event dispatcher being coupled to the interruption unit for dispatching the event transaction again to a processing unit.

A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for scheduling event transactions each having a respective time stamp, for use in an event-driven application having predetermined event rules, said method comprising: resolving possible time dependencies among events according to said predeter□mined event rules; and determining when the event transactions may be processed thus allowing the event transactions to be dispatched to distributed processors for concurrent processing.

A computer program product comprising a computer useable medium having computer readable program code embodied therein for scheduling event transactions each having a respective time stamp, for use in an event-driven application having predetermined event rules, said computer program product comprising: computer readable program code for causing the computer to resolve possible time dependencies among events according to said predeter□mined event rules; and computer readable program code for causing the computer to determine when the event transactions may be processed thus allowing the event transactions to be dispatched to distributed processors for concurrent processing.

The program storage device according to claim 9, wherein determining when the event transactions may be processed comprises: registering each incoming event so as to maintain a list of registered events; for each registered event resolving possible time dependencies among registered events and also with marked situations that are not detected yet according to said predeter□mined event rules; dispatching each event transaction for which all dependencies are resolved to an event processing unit; receiving a processing result of the event transaction; determining whether said event transaction is to be committed or aborted; and if the event transaction is to be aborted, aborting the event transaction and dispatching it again to a processing unit.

The computer program product according to claim 10, wherein the computer readable program code for causing the computer to determine when the event transactions may be processed comprises: computer readable program code for causing the computer to register each incoming event so as to maintain a list of registered events; computer readable program code for causing the computer to resolve possible time dependencies among registered events and also with marked situations that are not detected yet according to said predeter□mined

event rules for each registered event; computer readable program code for causing the computer to dispatch each event transaction for which all dependencies are resolved to an event processing unit; computer readable program code for causing the computer to receive a processing result of the event transaction; computer readable program code for causing the computer to determine whether said event transaction is to be committed or aborted; and computer readable program code responsive to the event transaction needing to be aborted for causing the computer to abort the event transaction and dispatching it again to a processing unit.

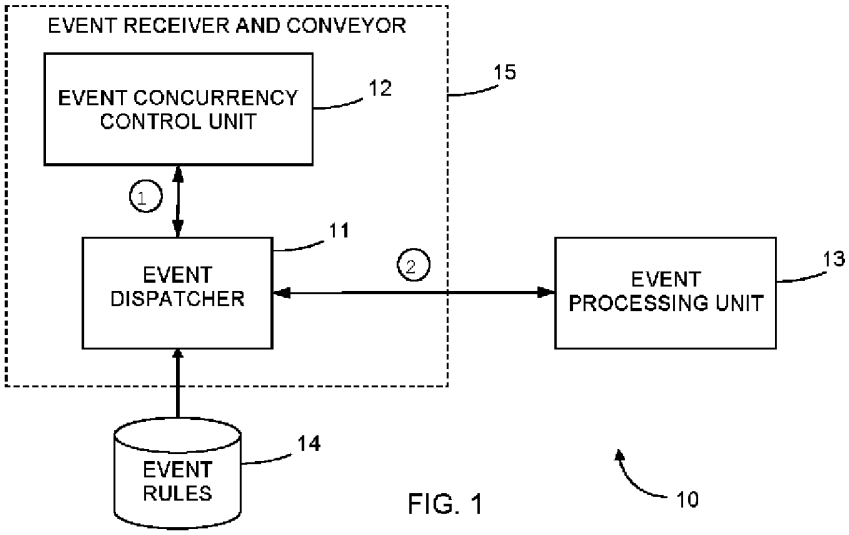

EVENT RECEIVER AND CONVEYOR
EVENT CONCURRENCY CONTROL UNIT
12
15
1
11
EVENT DISPATCHER
2
EVENT PROCESSING UNIT
13
EVENT RULES
14
10

FIG. 1

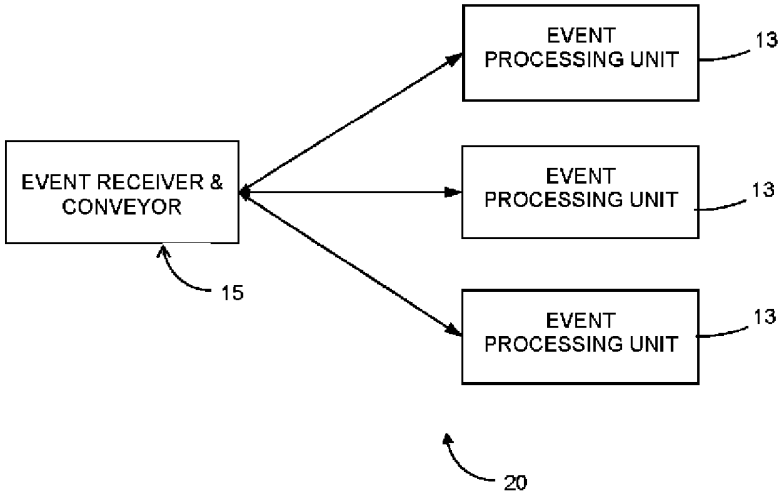

EVENT PROCESSING UNIT
13
EVENT RECEIVER & CONVEYOR
EVENT PROCESSING UNIT
13
15
EVENT PROCESSING UNIT
13
20

FIG. 2

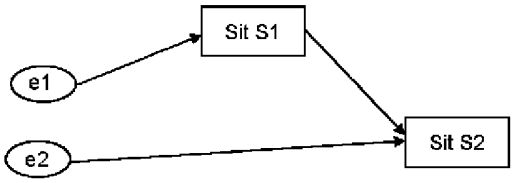

Sit S1
e1
e2
Sit S2

FIG. 3

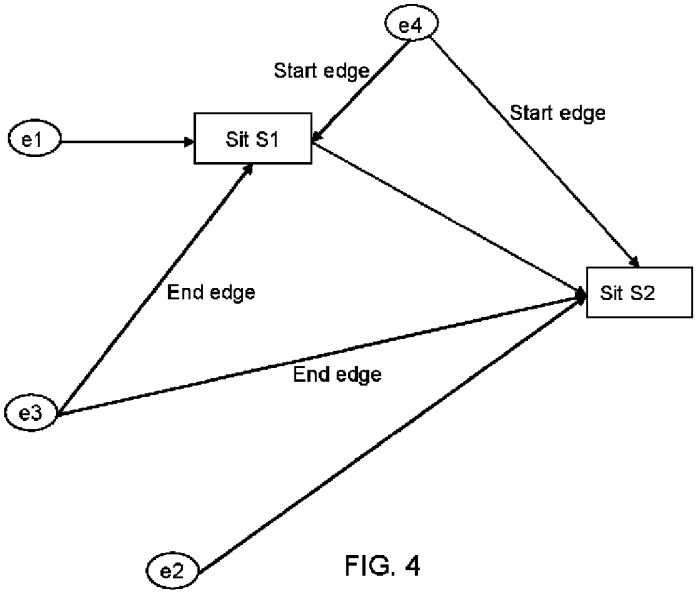

e4
Start edge
Start edge
e1
Sit S1
End edge
Sit S2
End edge
e3
e2

FIG. 4

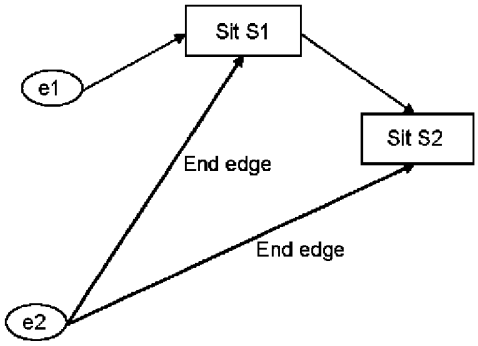


FIG. 5

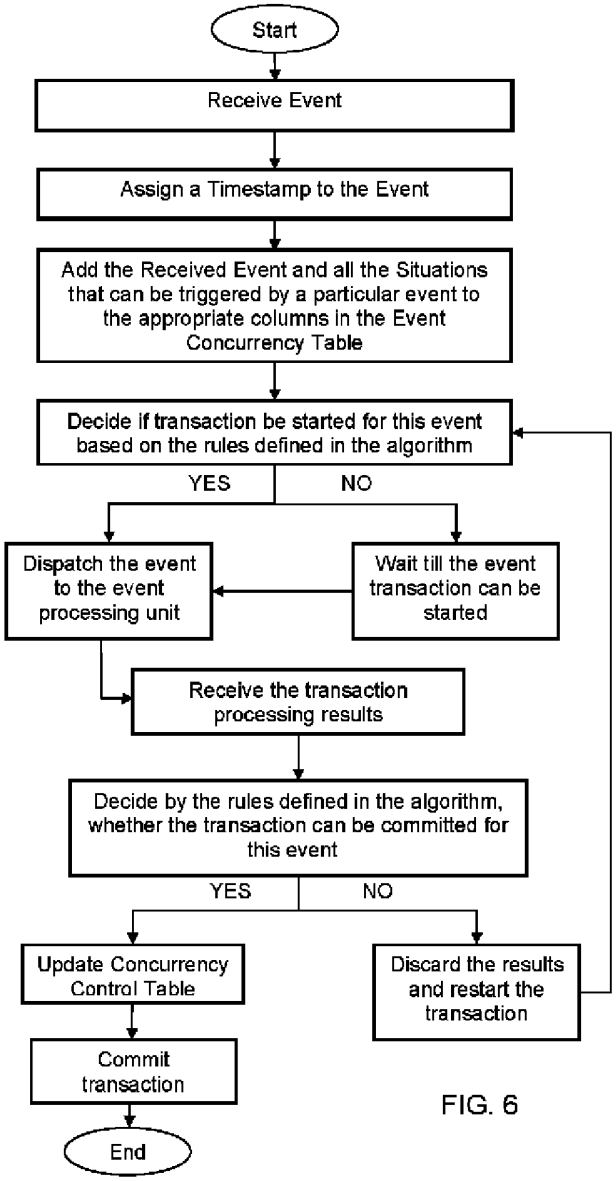

Start
Receive Event
Assign a Timestamp to the Event
Add the Received Event and all the Situations that can be triggered by a particular event to the appropriate columns in the Event Concurrency Table
Decide if transaction be started for this event based on the rules defined in the algorithm
YES
NO
Dispatch the event to the event processing unit
Wait till the event transaction can be started
Receive the transaction processing results
Decide by the rules defined in the algorithm, whether the transaction can be committed for this event
YES
NO
Update Concurrency Control Table
Discard the results and restart the transaction
Commit transaction
End

FIG. 6

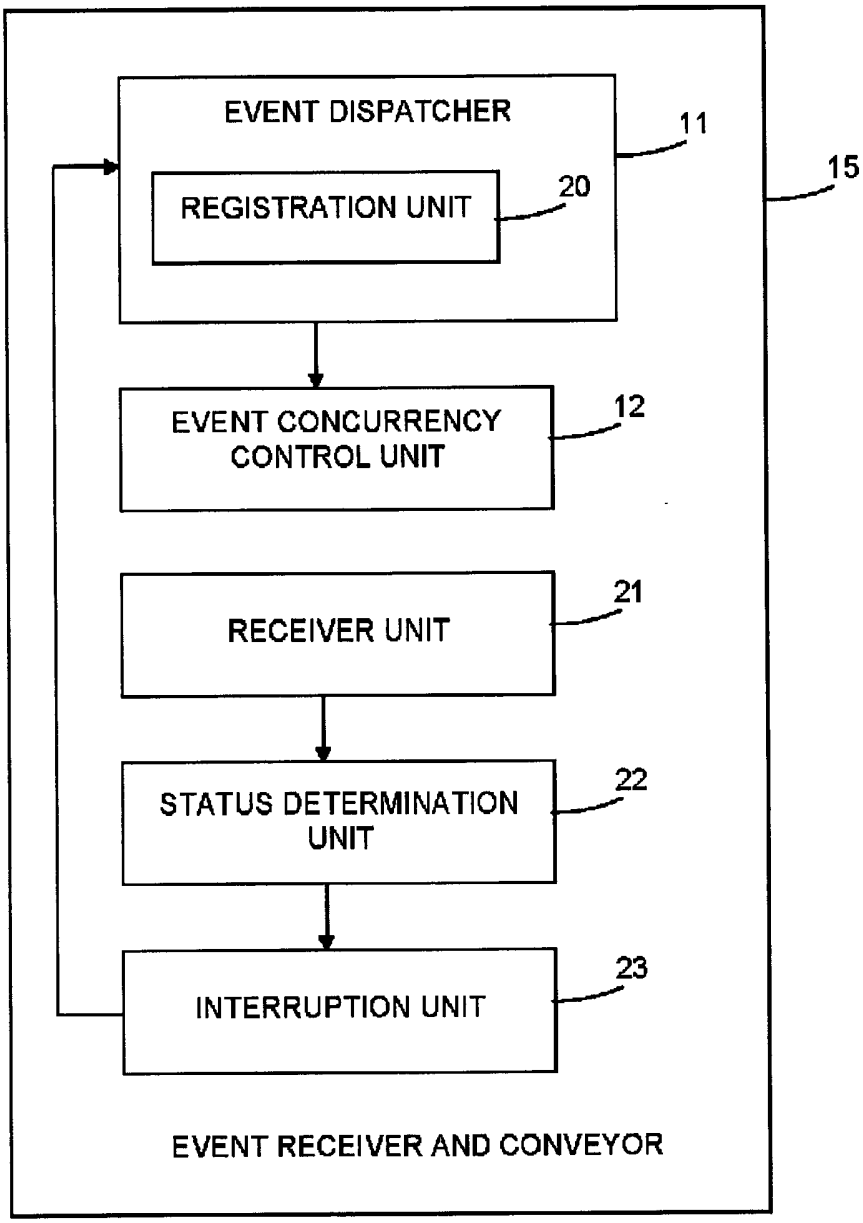

EVENT DISPATCHER
11
15
REGISTRATION UNIT
20
EVENT CONCURRENCY CONTROL UNIT
12
RECEIVER UNIT
21
STATUS DETERMINATION UNIT
22
INTERRUPTION UNIT
23
EVENT RECEIVER AND CONVEYOR

FIG. 7

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/052998

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/46

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)
G06F

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	BERNSTEIN P A ET AL: "TIMESTAMP-BASED ALGORITHMS FOR CONCURRENCY CONTROL IN DISTRIBUTED DATABASE SYSTEMS" PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES. MONTREAL, OCTOBER 1-3 1980, NEW YORK, I.E.E.E, US, vol. CONF. 6, 1980, pages 285-300, XP000866247 page 290, paragraph 4.1 - page 294, paragraph 4.8	1-12
Y	US 6 604 093 B1 (ETZION OPHER ET AL) 5 August 2003 (2003-08-05) cited in the application column 8, line 7 - column 12, line 26 column 13, line 36 - column 16, line 16	1-12

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but published on or after the international filing date

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 December 2005	20/12/2005

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Kusnierczak, P

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/052998

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	D. GEORGAKOPOULOS ET AL..: "Chronological Scheduling of Transactions with Temporal Dependencies" VLDB JOURNAL, 'Online! vol. 3, no. 1, 1994, pages 1-28, XP002357864 Secaucus, NJ, USA Retrieved from the Internet: URL:http://portal.acm.org/citation.cfm?id=615186> 'retrieved on 2005-12-05! the whole document	1-12
A	US 6 272 515 B1 (FOUQUET GUY) 7 August 2001 (2001-08-07) cited in the application column 2, line 55 - column 4, line 60	1-12

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP2005/052998

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6604093 B1	05-08-2003	NONE	
US 6272515 B1	07-08-2001	CA 2209697 A1	11-01-1998
		EP 0818730 A1	14-01-1998
		FR 2751106 A1	16-01-1998
		JP 10078876 A	24-03-1998