Complex event processing in enterprise information systems based on RFID

Chuanzhen Zang, Yushun Fan

National CIMS ERC (Engineering and Research Center), Department of Automation, Tsinghua University, Beijing, China, 100084, E-mail: zangcz03@mails.thu.edu.cn

Abstract Enterprises have to be increasingly agile and responsive to address the challenges posed by the fast moving market. With the software architecture evolving into SOA, and the adoption of RFID, event processing can fit well in enterprise information systems in terms of facilitation of event aggregation into high level actionable information, and event response to improve the responsiveness. To make it more applicable, the architecture of event processing in enterprise information system is proposed; event meta model and context serve as the solid basis for event processing; the rules, operators and keys of complex event processing are defined. Especially, workflow model is firstly used to extract complex event pattern. We have implemented the event processing mechanism in enterprise information systems based on RFID, including the architecture, data structures, optimization strategies and algorithm. The performance evaluations show that the method is effective in terms of scalability and the capability of event processing. Complex event processing can improve operational performance and discover more actionable information, which is justified by application. Finally, lessons learned from the application are presented.

Keywords Enterprise Architecture, Event driven architecture, Complex event processing, Event meta model, RFID, Work Flow, Application Integration, Service-Oriented Architecture, Manufacturing, Enterprise Software Systems

1 Introduction

The fast moving market makes it inevitable for enterprises to be more agile and responsive. To address these challenges, available information systems such as ERP, SCM, CRM, MES, etc. have improved the operational performance of enterprises in some extent. In addition, solutions like on demand enterprise, sense and respond enterprise, and real time enterprise aim at the improvement of agility and responsiveness.

Only when they are supported by effective architectural styles are these information systems and solutions possible (Fan et al., 2005). Service-Oriented Architecture (SOA) and Event-Driven Architecture (EDA) tend to be dominant in the architectural area. They both aim at maximizing the reuse of application-neutral components that increase IT adaptability and efficiency, and serve as key contributions to the evolving process of business integration (Jean-Louis, 2006).

Although these breakthroughs in information systems, solutions and software architectures work well in some extent, there are still some disadvantages.

1. There is still an information gap between the computation environment and the physical situation. It is difficult to automatically retrieve real time, accurate and detailed data such as what the accurate inventory volume is or where the parts are.
2. Even though the data can be retrieved, there is no effective mechanism to discover the business level, actionable information behind all these low level data. A case in point is that a credit card is used in Beijing at 9:21 AM, and after a few minutes there is a transaction of the same card in New York. An

effective method that can tell there is a card fraud and take some measures to prevent this fraud is substantially needed.

With the application of RFID (Radio Frequency Identification) (Nath et al., 2006), it is very possible to integrate virtual world and physical world to bridge the information gap. RFID can play an essential role in object tracking and supply chain management. In addition, it holds the promise to improve the operational performance in terms of inventory management, manufacturing process automation, even the personnel management. The data of physical world in enterprises can be read in bulky manner by elaborately deployed readers without line of sight, and then be processed by edge server (RFID middleware) and enterprise information systems, when the items, equipments and people are attached by RFID tags. Such RFID enabled enterprise environment has been piloted world wide, and it is very possible to be rolled out in the near future.

But the volume of data generated by RFID enabled systems can be enormous, which is beyond the processing capabilities of available information systems (Christof et al., 2005). A new mechanism is needed to process the enormous volume data as well as to discover the business level, actionable information behind all these data, and CEP (Complex event processing) (David, 2002) is a good candidate. The key ideas of CEP are as follows: first, primitive event extraction from large volume data; second, event correlation or event aggregation to create business event with event operators according to specific rules; third, event processing of primitive or composite event to extract their time, causal, hierarchical and other semantic relationships; and fourth, response to the actionable business information because of guaranteed delivery of events to the subscribers, and this is the event driven story.

The paper discusses where event processing can fit in the enterprise information systems enabled by RFID. The meta model, context, rules, operators and keys of event processing are formulated in detail. Especially, workflow model is firstly used to extract complex event patterns. System implementation and optimization strategies are also proposed specifically. The method and implementation are justified by performance evaluation as well as application in refrigerator manufacturing plant.

The remainder of this paper is organized as follows: in section 2, related works are discussed. In section 3, the architecture of event processing in enterprise information systems is put forth and discussed in detail. Complex event processing language is described in section 4, followed by section 5, namely, workflow model transformation into complex event pattern. The system implementation is proposed in section 6 followed by performance evaluation in section 7. Section 8 gives how complex event processing works in manufacturing enterprise. Conclusion and future works are presented in section 9.

2 Related works

The adoption of RFID poses new challenges for data processing and management. Major IT vendors have provided sensor solutions to address the challenges, such as Sun EPC Network (Gupta and Srivastava, 2005), SAP Auto-ID Infrastructure (Christof et al., 2005), Oracle Sensor Edge Server (Oracle, 2005), IBM WebSphere RFID Premises Server (IBM, 2005). These platforms collect data from sensors and RFID tags, process them with relative simple rules with the aim at filtering and duplicate removing, and then forward them to applications. Thus the applications need more efforts to find out more valuable, actionable, and business oriented information behind all these primitive and low level data.

To discover more valuable, actionable business level information, complex event processing is a good candidate. In fact, many research initiatives in the past have looked at event processing in active database research community (Stella and Klaus, 1994) (Liu et al., 1998). They are normally use ECA rules for event processing, and are difficult to support the RFID data processing which is characterized by temporal constraints.

In the more recent, there are some works that take advantage of complex event processing to process the RFID data, such as (Wang et el., 2006). But they lack implementation details and optimization strategies.

SOA (Jean-Louis, 2006) is helpful to maximize the reuse of application-neutral service to increase IT adaptability and efficiency, but it is based on conventional request/reply mechanism and one-to-one communications. Contrarily, EDA uses messaging to communicate and has advantages over SOA in terms of decoupled interactions, many-to-many communications, complex event processing that derives high level actionable information from massive primitive events.

3 Event processing in enterprise information systems

We argue that event processing will serve as a key player in building a flexible and responsive business when used either independently or together with SOA.

As a matter of fact, event processing is not new, and can be found over the history of computing (Opher, 2005), from exception handling in programming languages to active database, from network and system management to business activity monitoring. In these research areas, events are typically captured in dedicated applications or run on their own, without significant oversight. But in event driven applications, it is another story, where event serves as one of the core constructs in a well constructed architecture. It is very possible for event driven programming to evolve from being used at some niches to the main stream of application (Opher, 2005).

Application architecture of event processing in enterprise information systems we are working on is illustrated in figure 1.

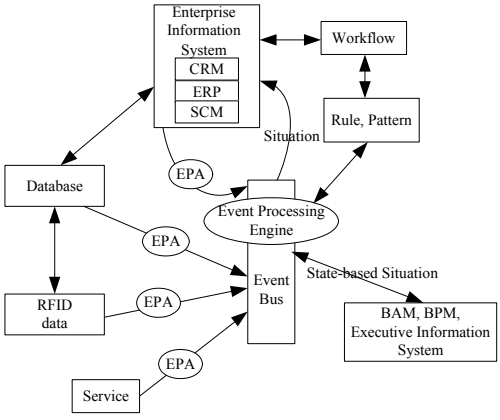

Fig. 1 Architecture of event processing in enterprise information system

In the daily business operations, millions of events can be produced by employees, customers, and suppliers by virtue of enterprise information systems, database and RFID. These events typically include data and messages that record activities. Specifically, enterprise information systems such as ERP, SCM, CRM and MES emit many events that are records of activities; insert, update and delete of data entries in database can also be viewed as events; the majority of events come from the extraction

of massive RFID data when the objects in enterprises are identified by RFID tag in pallet, case or item level; all kinds of services in SOA can also emit events. In addition, events can also be intercepted from message middleware.

These events are preprocessed by EPA (Event processing agent) in terms of filtering duplicate events, rectifying errors, format match, etc.

After being processed by EPA, these events are transferred into event bus, where events are aggregated into complex pattern according to certain rules, and then are sent to the subscribers in a guaranteed manner. The detected complex event or situation can then drive other activities in information systems. The rules and patterns are predefined though modeling tools, and they can be extracted from workflow models in some extent.

The architecture is an EDA story, which takes advantage of events, event aggregation and event subscription to improve the agility and responsiveness.

4 Complex event processing

In this section, complex event processing is discussed in detail. First, the formal definition of event is put forth; and event meta model is formulated, which serve as the solid basis for event processing. Second, event context, one of the core constructs, is proposed to provide more context information for event aggregation. Third, the syntax and semantics of constructs in our event processing language are presented, including complex event pattern, operators and keys.

Any information that users and systems are interested in can be defined as an event, such as state transition, activity record, etc. Event type and instance are generally represented by upper and lower case letters respectively, such as E and e.

Definition 1: event type E can be defined as E=(id, a, c, t_b, t_e). Each event type is uniquely identified by id; a={$attr_1$, $attr_2$, …, $attr_n$}, n≥0, is the attribute set that characterizes the event type; c={e_1,e_2, …, e_n}, n≥0, is the causality vector which contains the causal events that caused this event type to happen. Causality vector facilitates the behavior analysis for distributed systems. t_b, t_e is the starting and ending time respectively.

4.1 Event meta model

Events are not isolated, and they are interrelated with other constructs of information systems, such as event operators, context and process, which is illustrated by figure 2. The meta model of event proposed here presents the intrinsic relationships between different constructs in a formal way, which serves as the solid basis for event processing.

Events can be extracted from services, database, RFID and activities. Events are generally categorized into primitive events and complex events, both of which are characterized by properties, and there are causality relationships between events. Operators combine events together to form complex events or situations. These operators include logical operators, time operators, causality operators and RFID operators.

Event context is necessary for events to be aggregated into high level, actionable complex event. Context information includes semantic space, abstraction hierarchy and workflow model.

Semantic space can take events as its initiator and terminator; complex event happens in certain semantic space. Keys are defined according to event property and semantic space. Semantic space and key are discussed in section 4.2, 4.3.2 respectively.

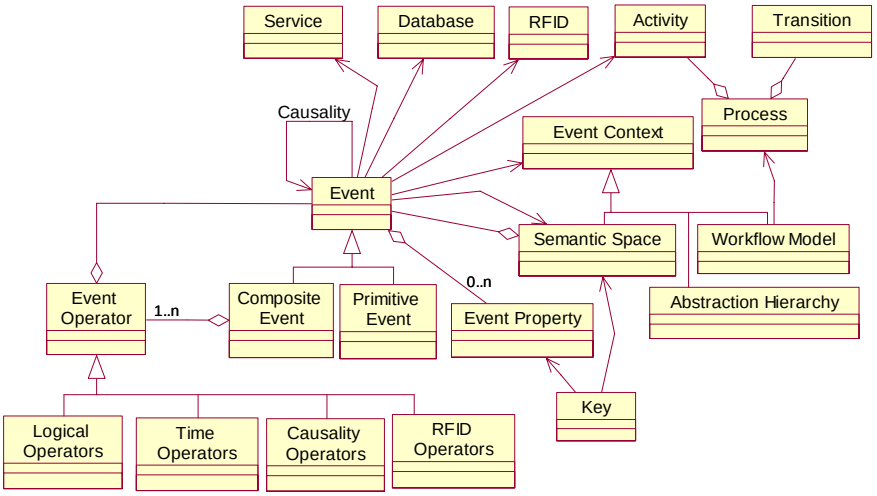

Fig. 2 Event meta model

4.2 Event context

RFID events and other extracted primitive events are low level and need context to be aggregated into actionable business information. Here, event context is used to denote any information needed to transform the low level events to high level information. Event context contains such elements as semantic space, workflow model, abstraction hierarchy of different dimensions eg. product, time, etc.

Semantic space is a relatively independent context of events, which is bounded by two events called initiator and terminator. The occurrence of an initiator event initiates the semantic space, and the occurrence of terminator event terminates it. In addition, it includes location, role, state and other relatively independent context information. Semantic space is an extension of lifespan in Amit (Asaf and Opher, 2004). An example of semantic space is shown in figure 3. Only the condition defined by "condition" holds can the initiator event initiate the semantic space. The type defined by "type" in the definition of terminator event means that the complex event related to this semantic space is to be discarded if it has not been reported and the semantic space will be terminated. There is another alternative of "type", namely, "deferred" which means that the semantic space will be terminated after all the unreported complex events have been reported according to the available event instances and context.

```
<SemanticSpace name="work shift">
   <initiator event=" shift transition">
      <condition name="group=group01" />
   </initiator >
   <location name="shop floor" />
   <role name="worker" />
   <state name="production" />
   <terminator event="shift transition">
     <type name="discard" />
   </terminator >
</SemanticSpace>
```

Fig. 3 An example of semantic space

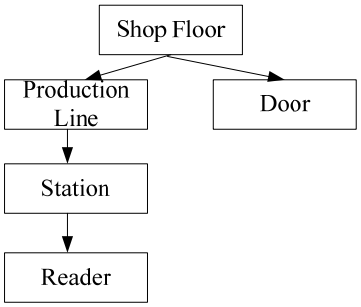

Fig. 4 An example of abstraction hierarchy

To be process compliance, workflow model is necessary to be transformed into event context. The information in the workflow model can be used to route the flow of items identified by RFID Tags, to check if the actual work route follows the model. An example of workflow in context is shown in figure 15.

Just like the data tube in data warehouse, an item can have a set of dimensions describing its characteristics, eg. product, brand, manufacturer. Each of these dimensions has an associated concept hierarchy (Gonzalez et al., 2006). Figure 4 presents the different levels at which a single item may be looked at along the location dimension. RFID reader is in the lowest level, and shop floor is in the highest level. When such concept hierarchies are carried by context, it will facilitate event aggregation in terms of attribute abstraction and semantic aggregation.

4.3 Rules for complex event processing

Definition 2: complex event processing rule is defined as "EVENT <complex event pattern> IF <qualification> DO <action>".

The key word "EVENT" denotes the complex event pattern. The action denoted by "DO" will be activated if conditions represented by "IF" hold.

Definition 3: complex event pattern is composed by operands and operators. Its definition is "complex event pattern={EACH} operator (operand {(con)}, …) {WHERE [equivalence test], parameterized predicates} {CONTEXT context {KEY key} {MODE mode}} {WITHIN, INTERVAL, AT}". For example: EACH SEQ (IN_FOAM_ROOM (reader="05AE") x, OUT_FOAM_ROOM y) WHERE [refrigerator-id] and x.weight<y.weight CONTEXT work shift MODE immediate WITHIN 10 min.

The brace "{}" denotes the optional item. "EACH" means that every instance of the complex event should be reported; otherwise, only the first complex event instance is reported.

"operand" refers to event type that is logically combined together by "operator" to create a new kind of event type. The "con" is used to retrieve specific event instance, such as (reader="05AE").

"WHERE" denotes the constraint conditions between operands, such as equivalence test and parameterized predicates. Equivalence test refers to the common attributes of all the operands, for example [refrigerator-id] means x. refrigerator-id＝y. refrigerator-id. Parameterized predicates are used to restrict different operands, such as x.weight<y.weight.

"CONTEXT" specifies the context information needed to detect the complex event.

Key is used for partitioning event instances, which will be discussed in section 4.3.2.

"MODE" denotes what to do when all the candidate event instances of complex event are available. It can be reported immediately or in deferred manner, represented by "Immediate" and "Deffered" respectively.

KEY and MODE will not be valid unless context is specified.

"WITHIN" denotes the time length of the complex event, "INTERVAL" denotes the specific time interval, and "AT" denotes the specific time point when the complex event happens.

In the following part, the syntax and semantics of operators and keys are discussed in detail.

4.3.1 Operator definition

The definition of operator requires the following items, as shown in figure 5.

1. The capacity of the operands linked by the operator. When the number of the instance exceeds the capacity, the oldest must be deleted according to aging strategy. The aging strategy is the rules that sort event instances according to specific attribute such as occurrence time of event, lead time of product, etc.
2. The using and consuming strategy of event instances. Complex event is defined with event type, and each event type can have many instances. The using strategy is used to decide which instance can be selected as the candidate component instance. When the instance is processed, it will be either deleted or preserved according to the consuming strategy. The possible values of using and consuming strategy are the same as Amit.

```
<operator name="AND">
    <length>50</length>
    <use>first</use>
    <consume>delete</consume>
</operator>
```

Fig.5. The definition of an event operator

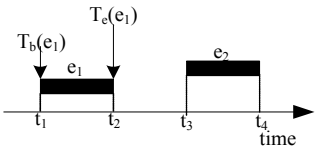

Fig.6. The illustration of point time, interval time and time operators

The commonly used event operator are AND, OR, NOT and SEQ, which take the same syntax and semantic as described in (Wang et el., 2006). In addition, there are other operators in terms of time, causality and RFID.

Ⅰ. Time operators

Event time can be categorized into interval time and point time, as shown in figure 6. Given e_1, its time is t_2 in terms of point time, and $[t_1, t_2]$ in terms of interval time.

Definition 4: time operator T(e) denotes the time of event e, $T_b(e)$ denotes the starting time and $T_e(e)$ denotes the ending time.

Generally speaking, primitive event uses point time and composite event uses interval time.

Definition 5: strong time relation, $T(e_1) > T(e_2) \Leftrightarrow T_e(e_1) > T_b(e_2)$, which means that the time intervals of different events are mutually independent.

Definition 6: weak time relation, $T(e_1) \succ T(e_2) \Leftrightarrow T_b(e_1) > T_b(e_2) \ \wedge T_e(e_1) \ < T_b(e_2)$. It means that the time intervals of different events are overlapped.

Ⅱ. Causality operators

The dependency between events can be characterized by causality relation.

Definition 7: immediacy causality operator is $\rightarrow$, namely, $e_1 \rightarrow e_2 \Rightarrow e_1 \in e_2.c$. It means that there are no intermediate events between causal event and result event.

Definition 8: intermediate causality operator is $\xrightarrow{\bullet}$, namely, $e_1 \xrightarrow{\bullet} e_2 \Rightarrow \exists e_i, i \neq 1,2, e_1 \rightarrow e_i \wedge e_i \rightarrow e_2$. It means that there are intermediate events between causal event and result event.

There is implicit consistency between time and causality, as explained by axiom 1 (David, 2002).

Axiom 1: $e_1 \rightarrow e_2 \vee e_1 \xrightarrow{\bullet} e_2 \Rightarrow T_b(e_1) \ > T_b(e_2)$. The starting time of causal event must be earlier than that of result event, irrespective of immediacy causality or intermediate causality.

Ⅲ. RFID Operators

In terms of RFID event, the primitive event extracted from massive data is in such form as PRIM (EPC, Loc, T) after the removing of duplicate and noise data (Bai et al., 2006). EPC is the unique identifier read by an RFID reader, “Loc” is the place where the RFID reader scans the item, and “T” is the time when the reading takes place (Gonzalez et al., 2006).

Definition 9: data compression operator, DCMPR(PRIM)=cmprData(EPC, Loc, T_1, T_2). It means that the object represented by EPC stays at location "Loc" during the time between T_1 and T_2.

Definition 10: path compression operator, PCMPR (cmprData)=cmprPath(EPC, (L_1, D_1), …, (L_n, D_n)). (L_1, D_1), …, (L_n, D_n) can be viewed as a path traveled by the item identified by EPC, where L_i is the i-th location in the path, and D_i is the total time that the item stays at the location.

Definition 11: group compression operator, GCMPR(PRIM)=cmprGroup(gid, Loc, T). In RFID applications, items tend to move and stay together in bulky manner (Gonzalez et al., 2006). "gid" is a generalized id which points to the only bulk, rather the original EPCs. By doing that, it can save substantial storage space.

4.3.2 The definition of keys for the partition of event instances

The candidate event instances are classified and partitioned by means of keys. Keys are used to match different instances that refer to the same entity; a key denotes a semantic equivalence among attributes that belong to different events (Asaf and Opher, 2004).

Keys together with semantic space can partition the event instances in different levels. A global key partitions the semantic space, and a local key divides all instances in a global partition into different groups, which is illustrated by figure 7.

Definition 12: global key={attr |attr ∈ situaton.operand.a and attr ∈ situation. semanticspace.initiator.a and a∈situation. semanticspace.terminator.a}. A global key is the common attributes of operands, initiator and terminator of semantic space. Here situation means complex event pattern and operand means component event type.

Definition 13: local key＝{attr | attr ∈situation.operand.a}. A local key is the common attributes of operands. Here situation and operand take the same meaning as the above definition.

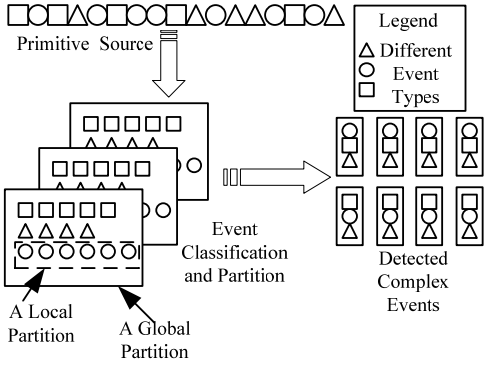

Fig. 7. Classification and partition of event instances

In general, the mechanism of event processing proposed here is different from Amit and Esper in that Amit has no declarative language for ease of use, no causal or RFID operators, and its context contains no workflow and abstraction hierarchy; Esper has no RFID operators or context that contains workflow model.

5 Workflow transformations into complex event pattern

The construction of complex event pattern is complicated and time consuming. Traditionally, when an activity in enterprise information systems finishes, it is typically mute and emits nothing to facilitate the response to event.

The information needed to construct complex event pattern are almost contained in workflow models, including events, constraints and operators. So, the following rules are given to facilitate the construction of complex event patterns. These rules are based on our application scenario in section 8, and rules in more formal manner are our future works.

1. Activity event based on location.

It is assumed that each activity is operated in one station in the production line, thus every activity is represented by a location.

Rule 1: the condition used to limit the PRIM event is extracted from activity property, and the name of complex event takes the name of the activity. For example, Condensation Assembly=PRIM (Loc=r_5), where r_5 is the RFID reader in this station.

2. Multi-readers are specified in an activity

When multi-readers are specified in an activity, there is implicit sequence relation between these events generated by the readers. For example, there are two readers in the "foaming" station, one of which is over the front door of foam room, and the other is over the back door. The extracted events are: IN_FOAM_ROOM=PRIM (Loc="r3"), OUT_FOAM_ROOM=PRIM (Loc="r4").

Rule 2: if there are multi readers specified in an activity, the complex event operator should be "SEQ", and the component events are extracted according to the physical location of readers. The time constraint in the complex event pattern is the same as the activity. For example, the above mentioned complex event is that FOAM = SEQ (IN_FOAM_ROOM, OUT_FOAM_ROOM) WITHIN 10min, where "10min" is obtained from the time constraint of "Foaming" activity in the workflow model.

3. Control logics such as split and join

There are many control logics in the workflow models, such as "and join", "or join", "and split", "or split", "iteration" , etc. These logics can also be transformed into complex event patterns according to the following rules.

Rule 3: if the control logic is "and join", the complex event operator will be "AND", and the component events will be extracted from the pre-activities. For example, from the "Synchronization" node we can extract such complex event as Sync=AND (Body, Liners Preinstall).

Rule 4: it the control logic is "and split", the complex event operator will be "AND", and the component events will be extracted from the post-activities. The time constraint of the complex event pattern is the maximum time constraint of all the post-activities.

6 Implementation

This paper has implemented the complex event mechanism. The core part is the architecture of event server that is illustrated by figure 8a.

The extractor is to extract event from enormous volume data generated by RFID, database, services and enterprise information systems.

Event receiving and publishing server is responsible for event type registration, subscription and publication. In addition, event instances are required to be saved and archived for post analysis. To make all these functions happen, the following interfaces are required.

1. Event registration interface: to register either primitive or composite event type.
2. Event subscription interface: to subscribe the interested event type.
3. Event publish interface: when the subscribed events are ready, they will be published by this interface, and then be delivered to subscribers in a guaranteed way, which is the "push" method.

4. Event query interface: users can query what they want deliberately, which is “pull” method.
5. Runtime interface: receiving event streams from extractor during runtime.

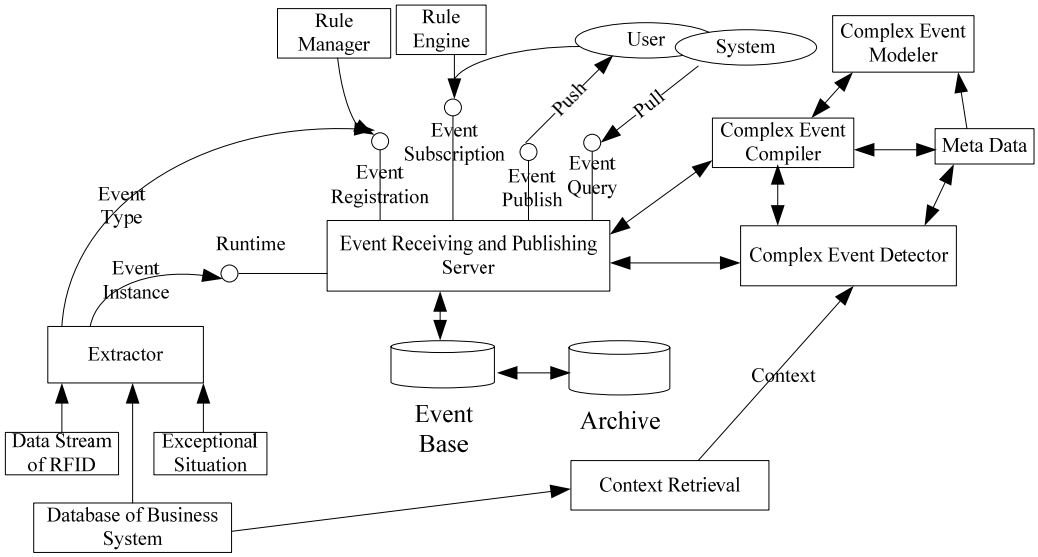

Fig.8a. The architecture of event server

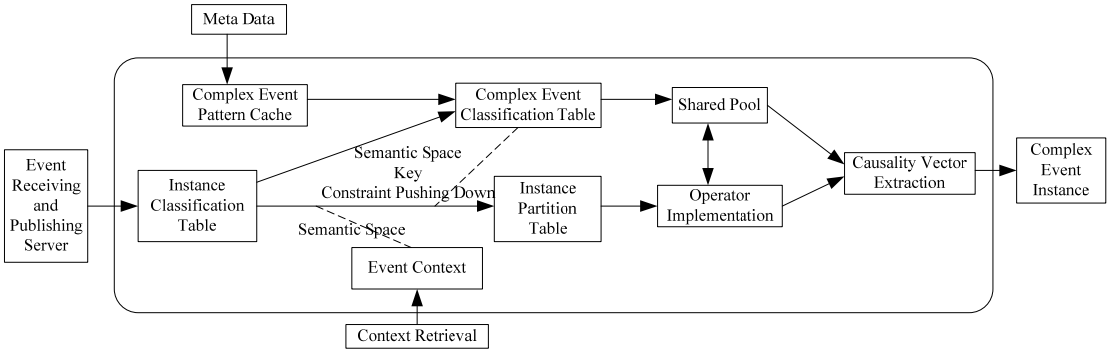

Fig.8b. The architecture of complex event detector

The main functions of complex event modeler are to create new composite event types and call the compiler to analyze them; to read, configure and maintain the meta information, such as using, consuming and aging strategies for operator definition; equivalence test, context, detection mode and keys for the definition of complex event pattern.

Meta data contain all the definitions of complex event patterns, each of which includes operator types, component event types, context, and time constraints. These meta data serve as the data dictionary for all the elements in the definition of complex event patterns.

The complex event compiler is to conduct syntactical and lexical analysis of complex event expression. Parser is used for parsing and walking of the complex pattern. Compiler is also responsible for checking if elements in the complex event pattern are valid according to meta data. If some elements referenced by the pattern are not available, an alert will be triggered.

The complex event detector is the core part of the architecture, as shown in figure 8b. It implements all the semantics of operators; provides shared pool of intermediate detection results and implements all kinds of optimization strategies which aim at improving the detection efficiency.

First, complex event pattern cache retrieves complex event pattern from meta data. These patterns are preprocessed for two purposes. One is to push down the constraints according to optimization strategies (section 6.2); the other is to construct complex event classification table (section 6.1).

Second, when event receiving and publishing server emits an event instance, the instance is classified into instance classification table. The detector finds out all the patterns which take the instance as component event according to complex event classification table; and then process the patterns one by one. In terms of each pattern, it checks if there is the same pattern that have been detected in the shared pool firstly. If yes, the detection result is utilized and the only thing to do is to substitute the causality vector, thus, substantial works will be saved; if no, the pattern has to be evaluated from the scratch.

All the instances that a pattern needs are retrieved from event instance classification table, and then are partitioned to form the instance partition table according to key and semantic space.

The detector takes advantage of instance partition table and operator implementation to evaluate if the complex event happens, where it can benefit a lot from the optimization strategies based on shared pool.

When a complex event instance is detected, it will be enriched by causality vector, and then be delivered to event receiving and publishing server.

6.1 Data structures

Complex event pattern is firstly compiled into a tree structure. For example, AND (RFIDEvent(reader=”05AE”), RFIDPartEvent), its tree structure is illustrated as figure 9.

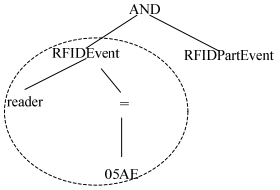

Fig.9. Tree for a complex event

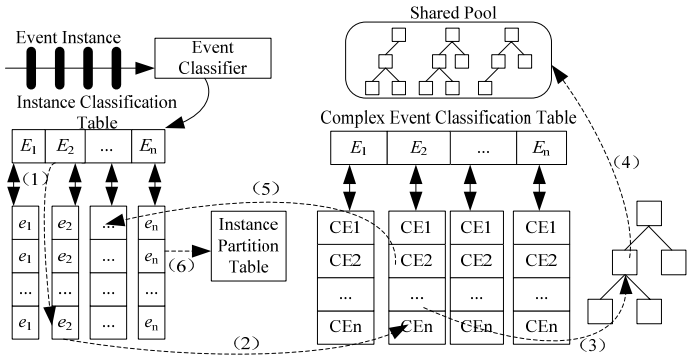

Fig. 10. Data structure for complex event detector

The data structure used for complex event detection contains primitive event classification table, complex event classification table, shared pool of intermediate result and component instance partition table, as illustrated by figure 10.

Instance classification table: each event type maintains an instance table of its own that has a configured capacity. If the number of event instance exceeds the capacity, the table will delete some instances according to aging strategies. These instances come from event receiving and publishing server.

Complex event classification table: each event type maintains a complex event classification table, which contains all the complex event expressions that take the event type as a component operand. When a complex event expression is compiled and verified to be correct, it will be saved to all its

operands' complex event classification table. For example, CE_1=AND(E_1, E_2), CE_2=AND(E_1, E_3), the complex event classification table of E_1 contains $CE_1$and CE_2; the table of E_2 contains only CE_1, and the table of E_3 contains only CE_2.

Shared pool of intermediate result: the intermediate detection results of each complex event are cached and saved for incremental detection, especially to share the detection results for different complex event expressions which have common sub-expression. And most of all, the shared pool provides basis for optimization of the complex event detection. For example, CE_1 = SEQ (AND (A_1, A_2), A_3)，CE_2 = OR (AND (A_1, A_2), A_3)，during the process of CE_1 detection, if AND(A_1,A_2) is firstly detected, it will be cached in the shared pool as an intermediate result. When the instance of A_3 arrives, the ultimate detection result can depend on the intermediate result and A_3, thus it is not necessary to detect AND (A_1, A_2) repeatedly. At the same time, the detection of CE_2 can also use shared pool to improve detection efficiency.

Instance partition table: all the candidate event instances that a complex event pattern requires are selected from instance classification table, and are partitioned to improve detection efficiency according to keys and semantic space. The partitioned event instances are contained in the component instance partition table.

6.2 Optimization strategies

Based on the above data structures, optimization strategies are provided to detect complex event effectively.

Ⅰ. Operator implicit semantics

Theorem 1: there is implicit semantic containment relation between operators SEQ, AND and OR, that is, $SEQ(E_1,...,E_n)\ WITHIN\ T \Rightarrow AND(E_1,...,E_n)\ \ WITHIN\ T \Rightarrow OR(E_1,...,E_n)\ WITHIN\ T$

Theorem 2: there is implicit semantic exclusive relation between SEQ (E_1, !E_2) WITHIN T and SEQ(E_1, E_2) WITHIN T. They can not happen at the same time.

Theorem 3: cmprPath (EPC, (L_1, D_1), …, (L_n, D_n))=PCMPR(PRIM) $\Rightarrow$ SEQ(PRIM (Loc_i=L_i)), where $0 < i \le n$. The sequence relationship between primitive RFID events can be inferred from path compression operator PCMPR.

Ⅱ. Constraints pushing down

In the process of complex event computation, there would be many intermediate results, some of which satisfy the constraints and others do not. To improve the detection efficiency, it is essential to reduce the intermediate results as soon as possible. An effective measure is to push down the complex event constraints and exclude the unrelated intermediate results as early as possible. For example, SEQ (AND (A, B), C) WITHIN 30 equivalents to SEQ (AND(A, B) WITHIN 30, C) WITHIN 30.

Another constraint that needs to be pushed down to reduce intermediate results is equivalence test, such as $AND(A,B)\ \ WHERE\ [attr_1, attr_2]$. It is necessary to reduce the number of candidates according to the equivalence attribute and partition the candidates to improve the detection efficiency.

Definition 14: if there are k equivalence attributes, and they have $i_1, i_2, ..., i_k$ partitions respectively, then the following sets are given:

$I_{1,1} = \{e \mid e.attr_1 = value_1\}$, ..., $I_{1,i_1} = \{e \mid e.attr_1 = value_{i_1}\}$, … , $I_{k,i_k} = \{e \mid e.attr_k = value_{i_k}\}$.

For example, in figure 11,

instance	a_1	a_2	b_3	a_4	b_5	b_6	a_7
attr1 value	1	2	1	2	2	1	1
attr2 value	3	2	2	3	2	3	2

partition	Attr1 =1	Attr1 =2	Attr2 =2	Attr2 =3
	a1	a2	a2	a1
	b3	a4	b3	a4
	b6	b5	b5	b6
	a7		a7	

Fig.11. Equivalence test constraint

$I_{1,1}=\{a_1,b_3,b_6,a_7\}$, $I_{1,2}=\{a_2,a_4,b_5\}$, $I_{2,1}=\{a_2,b_3,b_5,a_7\}$, $I_{2,2}=\{a_1,a_4,b_6\}$. Then, $R_{1,1}=I_{1,1}\bigcap I_{2,1}=\{b_3,a_7\}$, $R_{1,2}=I_{1,1}\bigcap I_{2,2}=\{a_1,b_6\}$, $R_{2,1}=I_{1,2}\bigcap I_{2,1}=\{a_2,b_5\}$, $R_{2,2}=I_{1,2}\bigcap I_{2,2}=\{a_4\}$. Thus, the detection result is $R_{1,1}, R_{1,2}, R_{2,1}$.

6.3 Algorithm

The complex event detection algorithm is illustrated by figure 12.

```
1.  PARSE(cmplx_expr)
2.  SAVE_RELATED_CMPLX_EVENT()
3.  PUSHING_DOWN_CONSTRAINTS()
4.  RECV_EVENT_INSTANCE()
5.  for each event instance e
6.    FIND_RELATED_CMPLX_EVENT()
7.      for each complex event expression cmplx_expr
8.        tree=TREE(cmplx_expr)
9.        t=FIND_IDENTICAL_TREE(tree)
10.       t=t∪FIND_SEMANTIC_CONTAINMENT(tree)
11.       t=t∪FIND_ SEMANTIC_EXCLUDE(tree)
12.       if t is null
13.         GET_CANDIDATE_LIST()
14.         if there is equivalence test
15.           PARTITION_CANDIDATE_LIST()
16.       end if
17.     REPORT_EVENT()
18.     SAVE_RESULT_TO_SHARE_POOL()
```

Fig.12. Complex event detection algorithm

Line 1-3 is the initialization works including syntactical and lexical analysis, writing the complex event classification table, and pushing down the constraints. Line 4-18 conduct the complex event detection algorithm. First, receive event instances and save them to instance classification table (line 4), and then look up the complex event classification table (line 6) for each instance. Each related complex event is evaluated to happen or not (Line 7-18). After the tree structure is obtained (line 8), if there is the same intermediate result in the shared pool (line 9), it is not necessary to conduct other detection procedures. And then, check if there is implicit semantic containment (line 10) and exclusive (line 11) complex event. If there are no such intermediate results, it is necessary to retrieve the candidates (line 12-16), where, these candidate instances are classified and partitioned by keys and semantic space specified in the definition of complex event pattern. During this process, if there is equivalence test, the instance candidates can be partitioned. The sub complex events are evaluated according to the operator semantics. If the result is true, it is reported and then cached in the shared pool.

The time complexity of this algorithm is O(nmk) in the worst case, where m, n, k is the number of complex event patterns, event instances and event types respectively. Because m and k are relatively fixed values and not very large in a system, the processing time is proportional to n.

In terms of implementation, Amit and Esper have no optimization strategies proposed here, including shared pool, operator implicit semantics and constraint pushing down. So better performances can be expected from our method.

7 Performance evaluations

The prototype is implemented in Java language, which name is RTE-CEP, and the test environment is as follows: Pentium IV 3.0 GHz CPU，1G memory.

There are 20 event types for performance test, that is, $E_1, E_2, ..., E_{20}$, each of which has simple attributes and methods to get their attributes. There are also 50 complex event expressions. To test the performance, the over-loaded test are used here, namely, each event type is used by at least one complex event expression that takes it as a component event. Each complex event contains AND(AND(E_1, E_4), AND (E_2, E_3)) to test the function of shared pool, that is, each of E_1, E_2, E_3, E_4 event type has 50 complex event expression in their complex event classification table.

Such environment for performance test is set for application in section 8. It is enough for our application scenario, and performance test in more complex environment is our future work.

The event instances are generated randomly. For each number of input events, 10 tests are conducted and its mean value is utilized for analysis.

To date, there is no generally accepted benchmark for complex event processing. The similar solutions are Amit, Esper, etc. Because most of these solutions such as Amit are embedded into other commercial software products, that is, they are neither independent products nor available.

Esper(Thomas, 2005) is an independent product, which aims at addressing the requirements of applications that analyze and react to events. It is built in complete accordance with the idea of CEP posed by David Luckham (David, 2002). And it is open source software. So Esper is used for performance comparison.

The comparison of event processing time is shown in figure 13. When the number of input event is less than 700000, RTE-CEP has a slight advantage over Esper; when the number exceeds 700000, the processing time of RTE-CEP is longer than that of Esper, the reason of which is that RTE-CEP use content incremental detection method and the number of detected complex events is much larger than Esper. The processing capacity per second of these two methods is similar, namely, about 6000/s in the test environment here.

From this figure, we can learn that the event processing time is proportional to the number of primitive event instances, as shown in the time complexity analysis of our algorithm.

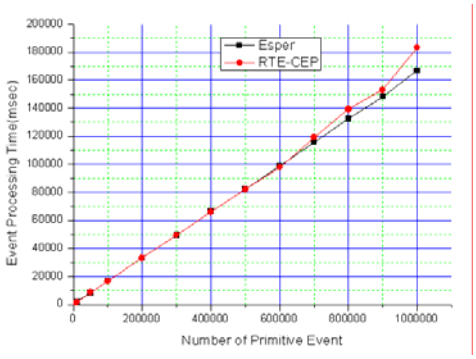

Fig.13. Event processing time of two methods

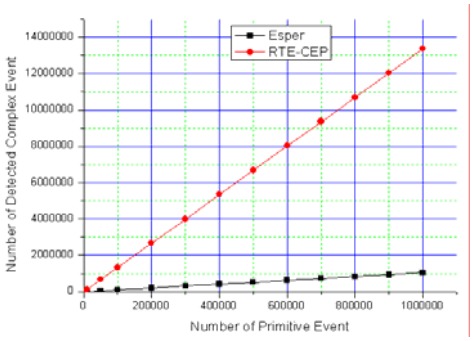

Fig.14. Number of detected complex events of two methods

Figure 14 shows the number of detected complex events of the two methods. RTE-CEP has apparent advantage over Esper because of the content incremental detection method of RTE-CEP. As for AND(E_1, E_4), there are such event sequences as $e_1^1, e_3^2, e_4^3, e_4^4$, where e_i^j means the event instance of event type E_i at j time point, RTE-CEP has detected two complex events((e_1^1, e_4^3),(e_1^1, e_4^4)), while Esper detected only one ((e_1^1, e_4^3)).

In terms of the number of detected complex event per second, there is apparent difference between these two methods. RTE-CEP method can generate 80000/s, but Esper can only generate 6000/s.

In general, our method is better than Esper in terms of capabilities of event processing because of our elaborated data structures and optimization strategies.

8 Applications

RFID is deployed in a major refrigerator manufacturer in China to meet the requirement of Wal-mart. And RTE-CEP is implemented to reap more from process reengineering based on RFID.

The simplified workflow model of the refrigerator manufacturing process is shown in figure 15. The workflow modeling tool used here can specify RFID reader in its resource property of activity. The body and liners are concurrently prepared. When they are both ready, the activities of body assembly, foaming, condensation assembly, foaming check and packaging are followed. The "Foaming Check" activity is a "or split" node, where if the foaming is qualified the next activity will be "Packaging" and if the foaming is not qualified the refrigerator must be refoamed.

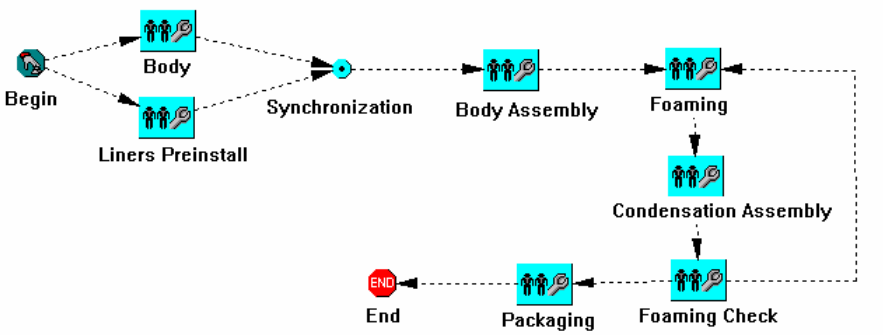

Fig. 15 An example of workflow model in refrigerator manufacturing plant

First, RTE-CEP is useful to address problems in operational level, such as mismatch between parts and refrigerator, inaccurate assessment of work time and quality, and product shipment error.

When a refrigerator is about to assemble a compressor in the station of “Condensation Assembly”, the types of refrigerators and compressors are identified by RFID reader. Complex event “A=AND (REFRIGERATOR, OR (COMPRESSOR01, COMPRESSOR02)) WHERE [compressor_type] INTERVAL 10 min” can assure the match between refrigerator and compressor. If A equals NULL, an alert will be activated to tell the worker that the wrong type of compressor is being assembled.

To assess work time and quality accurately, complex event “EACH SEQ (WORKTIME, QUALITY) WHERE [person_id, station] INTERVAL AUGUST” is used. The event can analyze work time and quality in every station of every person according to the accurate record of primitive event WORKTIME and QUALITY.

The product shipment error can be addressed by such complex event as “EACH SEQ (TRUCK, EXIT-READING (type != context. type))”, where TRUCK event refers to the fact that the vehicle is ready and event “EXIT-READING (type !=context.type)” assure that the right product is shipped to right customer.

Second, RTE-CEP is helpful in management level. It can facilitate the analysis of such concerns of executives as why refrigerators operated by the third group of workers tend to breakdown. The facilitation is achieved because event processing can maintain the causality vector of complex event, and the concerns can be drilled down to the event sources following the causality vector. In addition, the detected event can be delivered to the subscribers in real time manner, so managers can make quick decisions.

In general, CEP is critical for enterprise information systems based on RFID in the following aspects.

1. CEP can discover more high level, actionable information behind RFID data and other business data.
2. CEP can extract the time, causal and hierarchical relationships between events. It facilitates the analysis of system behaviors and can pinpoint where the root cause is.
3. CEP can improve the system responsiveness in terms of timely delivery of events.

From the application some lessons are learned. First, the adoption of RFID technologies and CEP is a long term of task and needs cross organizational collaboration. It is imperative to integrate RFID into business process and process reengineering is required sometimes to make RFID support process operation in a more effective and efficient way. Second, the wrong complex event is reported or the expected complex event is not reported from time to time. One of the reasons is that we can not expect a 100% reading accuracy because of the immature technology and harsh working environment. In addition, there are dead spots in the coverage areas of different readers. It can be overcome by the elaborated deployment of readers in the way that the coverage areas of different readers are overlapped in certain extent.

9 Conclusions and future work

Enterprises have to be increasingly agile and responsive to address challenges posed by fast moving market. We argue that with the evolvement of software architecture into SOA and the adoption of RFID, event processing can be an important player in enterprise information systems in that it is easy to construct decoupled, many to many communications systems, and it facilitates event aggregation to derive more actionable information.

In this paper, the description where event processing can fit in enterprise information systems is put forth. The event meta model and rules of complex event processing are formulated. The system implementation is discussed in detail. Performance evaluation and application show that event processing is effective to increase agility and responsiveness.

We will formulate the rules of complex event pattern transformation from workflow model in more formal manner.

References

Asaf A, Opher E. Amit-the situation manager. *VLDB* 2004; 13(2): 177-203.

Bai Y, Wang F, Liu P. Efficiently Filtering RFID Data Streams. In: *CleanDB* 2006.

Christof B, Tao L, Stephan H, Joachim S. Integrating Smart Items with Business Processes An Experience Report. In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)* 2005; 227-235.

David L. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison Wesley, 2002.

Fan Y, Huang C, Wang Y, Zhang L. Architecture and operational mechanisms of networked manufacturing integrated platform. *International Journal of Production Research* 2005; 43(12): 2615-2629.

Gupta A, Srivastava M. Developing Auto-ID Solutions using Sun Java System RFID Software. *SUN Corporation*, retrieved from http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/sjsrfid/RFID.html on November 21, 2005.

Gonzalez H, Han J, Li X. FlowCube: Constructing RFID FlowCubes for Multi-Dimensional Analysis of Commodity Flows. In: *VLDB* 2006.

IBM. IBM WebSphere. *IBM Corporation*, retrieved from http://www-306.ibm.com/software/integration/ on November 21, 2005.

Jean-Louis M. Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus. retrieved from http://www-128.ibm.com/developerworks/webservices/library/ws-soa-eda-esb/index.html#Resources on September 17, 2006.

Liu G, Aloysius KM, Prabhudev K. A unified approach for specifying timing constraints and composite events in active real-time database systems. In: *Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium* 1998; 199 - 208.

Nath B, Reynolds F, Want R. RFID technology and applications. *IEEE Pervasive Computing* 2006; 5(1): 22-24.

Oracle. Oracle Sensor Edge Server. *Oracle Corporation*, retrieved from http://www.oracle.com/technology/products/sensor_edge_server/index.html on November 21, 2005.

Opher E. Towards an Event-Driven Architecture: An Infrastructure for Event Processing Position Paper. In: *Proceedings of First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML)* 2005; 1-7.

Stella G, Klaus R. Detecting Composite Events in Active Database Systems Using Petri Nets. In: *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering*1994; 2-9.

Thomas B. Esper. retrieved from http://esper.codehaus.org on November 21, 2005.