Extended CEP Model for Effective Enterprise Systems Service Monitoring

Deuk Kyu Kum
Dept. of Information and Communication Engineering, Yuhan University
Bucheon-si, Gyeonggi-do, 14780, South Korea
[e-mail: dkkum1@yuhan.ac.kr]
*Corresponding author: Deuk Kyu Kum

Received Septebmer 30, 2019; revised December 16, 2019; accepted January 12, 2019; published February 29, 2020

Abstract

In recent years, business environments have become more complex; therefore, enterprises must be capable of responding flexibly and agilely. For these purposes, effective enterprise systems service monitoring and early decision making based on the same, emerge as core competency of the enterprise. In addition, enterprise system techniques that filter meaningful data are needed to event processing. However, the existing study related with this is nothing but discovering of service faults by monitoring depending upon API of BPEL engine or middleware, or is nothing but processing of simple events based on low-level events. Accordingly, there would be limitations to provide useful business information. In this study, we present an extended event processing model that enables delivery of more valuable and useful business information through situation detection. Primarily, the event processing architecture in an enterprise system is proposed as a definite approach, and then define an event meta-model suitable for the proposed architecture. Based on the defined model, we propose the syntax and semantics of the elements that make up the event processing language include various and progressive event operators, the rules, complex event pattern, etc. In addition, an event context mechanism is proposed to analyze more delicate events. Finally, the effectiveness and applicability of proposed approach is presented through a case study.

Keywords: Complex Event Processing (CEP), Enterprise Systems, Service Monitoring, Event Context, Event Driven Architecture (EDA)

A preliminary version of this paper was presented at APIC-IST 2019, and was selected as an outstanding paper.

http://doi.org/10.3837/tiis.2020.02.019 **ISSN : 1976-7277**

1. Introduction

Because of the complexity and competitiveness of business environments in recent years, enterprises today need to be more flexible and agile in their response. To address this challenge, enterprises have improved their operational performance to some extent using available enterprise systems such as customer relationship management (CRM), enterprise resource planning (ERP), and manufacturing execution system (MES) [1]. In addition, as more importance is given to early decision making, providing business-level, actionable information and quickly responding to changes through run-time service monitoring have become core competencies for enterprises. In this context, service-oriented architecture (SOA) and event-driven architecture (EDA) are attracting attention as solutions that support these enterprise systems [2, 3].

While SOA increases IT adaptability and efficiency by maximizing the reuse of application-independent services, typical communication methods such as request/response and synchronous communications are overloaded and can introduce latency, making them unsuitable within a monitoring environment where any generated events or exceptions are required to be processed in a timely manner. EDA on the other hand can address these problems by utilizing publish-and-subscribe and asynchronous communications; EDA also ensures agility and responsiveness and complements SOA by efficiently collecting and analyzing the various events that occur in enterprise systems [3, 4]. However, little research has been conducted on applying or tailoring EDA towards effective service monitoring. There is insufficient existing research to provide useful information at the business level, because monitoring based on APIs of the business process execution language (BPEL) [5] engine or middleware is not only detecting service defects or primitive event processing based on low-level events.

Complex event processing (CEP) is one EDA-based technology suitable for service monitoring; CEP analyzes the effects of events from various sources, provides meaningful information, and processes corresponding actions [3]. It can also provide a systematic method of implementation by using event-condition-action (ECA) rules [4, 6].

In this study, an extended CEP model is proposed to effectively handle various events occurring in a complex business environment. The proposed event processing model is then applied to service monitoring in an enterprise system. Primarily, an event processing architecture is proposed for an enterprise system as a definite approach, and then define an event meta-model suitable for the proposed architecture. Based on the defined model, the syntax and semantics of the constructs are written using a language, including various, advanced event operators, patterns and rules. In addition, we propose an event context mechanism for more delicate event analysis. Finally, the effectiveness and applicability of the proposed approach is presented through a case study.

2. Related Work

Existing service monitoring studies only utilize SOA-based technologies, and APIs of the BPEL engine or middleware are utilized. In other words, data was collected to calculate the quality metrics common to many existing quality models such as time, availability, reliability, and security; the range of data that can be collected in this manner is limited to what can be

obtained through external environments such as the enterprise service bus (ESB). Lin [7] suggested a middleware implementation, called Llama, that extends the ESB. This middleware implementation consists of one component for service monitoring, one component for detecting service defects, and one component for their diagnosis. Llama is a method of monitoring service execution times; it also demonstrates the acquisition of monitoring data by utilizing either the ESB monitoring API or additional components (such as Profiling Interceptors). Baresi [8] suggested an integrated monitoring framework that operates between Dynamo and Astro. Dynamo uses aspect-oriented programming (AOP) to obtain monitoring data that the BEPL engine collects while the BPEL process is running. Astro is a method of verifying attributes defined in a run-time monitor specification language (RTML) using independent software modules.

The above two studies only discuss the addressing of service defects, because the proposed methods used to acquire any sent and received messages are dependent on the BPEL engine and middleware. In addition, SOA's request and response, synchronous communication, and pooling interaction models generate significant communication overhead and are not suitable for monitoring where rapid processing of events or exceptions is required [3, 4, 6].

EDA delivers goods, services, and information in a way that guarantees minimal delay by implementing event exchange, event triggers, and real-time responses, based on various event situations within an integrated service-based information system architecture [9]. EDA is a publishing and subscribing, asynchronous communication, and push interaction model that has major advantages in service monitoring; short trip times, high throughput, and low communication load are examples of these [3, 4, 9]. In addition, the supplier provides information to the consumer who can subscribe or register to receive any necessary information first, thus enabling an early response to potential opportunities and threats [3, 4]. Techniques for handling events can be largely divided [6, 9] according to event handling methods listed below.

- Primitive event processing: All events that have occurred are regarded as meaningful and corresponding actions are performed according to the contents of each event. The event processing method is provided by publishing, subscribing, or mediating.

- Stream event processing: A large number of event streams are targeted in which both meaningful and meaningless events occur together. Only meaningful event information is filtered, extracted, and delivered to the application or service.

- Complex event processing: The composition and meaning of events are analyzed and meaningful and useful information for events from various sources is provided [9]. While primitive event processing targets one event, complex event processing analyzes the various relationships between different events.

McGregor proposed a solution manager service (SMS) architecture for measuring business process performance [10]. The proposed architecture supports web service logs and enables them to manage a large volume of process events in real time through a container. The advantage of the proposed architecture is that it integrates log data from business processes to identify potential quality attributes and provide comprehensive information to both suppliers and consumers to avoid delays in decision making. However, the event processing object is limited to primitive event processing for low-level events; therefore, there is a limit to how much useful business information can be provided to the enterprise.

Wang pointed out that the complex event definition method, which depends only on the type of event, is inherently ambiguous and that the event consumption model, a partial solution to this, is also not flexible [11]. As an alternative complex event definition method, the event model of Real-Time Logic (RTL) [12], which is used mainly in the specification of real-time systems, is extended to define complex event definition methods and related event operators based on event instances. However, because only the general ECA rules are used for event processing with a focus on a specific domain called a network management system, there is a limit to the capabilities of a cause analysis through refined situation detection.

3. CEP in Enterprise Systems

In this chapter, we propose an event processing architecture and application technique for enterprise systems and examine the distinctive features of the proposed architecture.

3.1 Enterprise event processing architecture

Event processing is not a new technology, but it has been used in specific areas such as active databases and network management systems. It has also been used independently without a holistic view of enterprise systems [3]. However, event processing can play a key role in providing more valuable, actionable business information and responses by effectively analyzing events occurring inside and outside the enterprise for service monitoring. **Fig. 1** shows a block diagram of the proposed enterprise event processing application architecture.

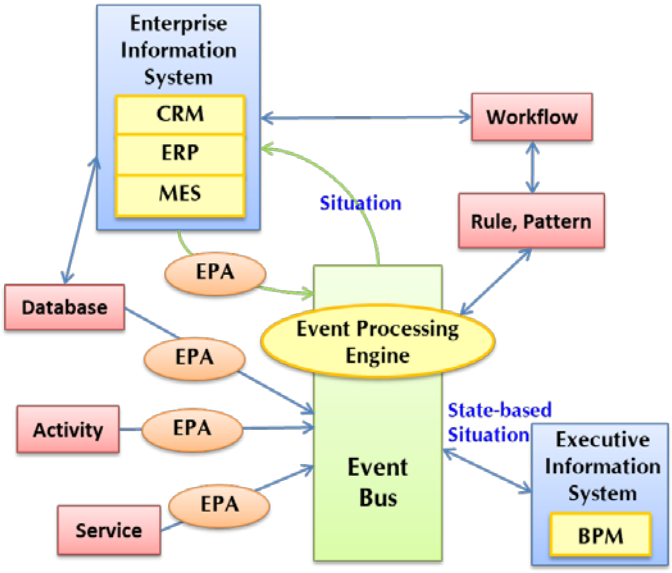

Fig. 1. Enterprise event processing architecture

Various business activities performed by suppliers, consumers, and customers can generate many events through services, activities, databases, and enterprise systems. These events typically contain data and messages about business activity, and enterprise information systems, such as CRM, ERP, and MES in particular, generate a large number of events that

correspond to the recording of activities [1]. Business process management (BPM) is an effective and integrated tool for business management and execution [13]. BPM perform business automation based on events that are generated through the execution of business unit activities that form logical steps within business processes [13, 14].

The event processing agent (EPA) collects the generated events, filters the overlapping events, and performs preprocessing such as matching an event format. The events preprocessed using the EPA are routed to the event bus to apply specific rules and patterns and then processed and refined through the processing of the event processing engine. Complex events or situations detected during event processing serve as information needed for decision making, and the rules and patterns can be defined in advance.

3.2 Characteristics of the proposed event processing architecture

The core functions of the event processing architecture can be defined in three ways: collecting numerous events in real time, analyzing the collected events, and responding appropriately to the events [3, 4, 6]. The proposed enterprise event processing architecture is designed to apply general event processing architecture functions to service monitoring in enterprise systems and provides the basis for the design of the complex event processing model, especially the event meta-model, which is proposed in Chapter 4. The distinguishing characteristics of the proposed architecture are as follows.

- Event collection: The EPA collects and processes various events occurring in an enterprise environment, which enables only the EPA to directly interact with event sources, improving the modularity of event sources and buses, and providing meaningful data through EPA's preprocessing process. You can effectively handle events by filtering them. In addition, the closest EPA responsible for each event source can collect events, increasing time efficiency.

- Event analysis: Events moved to the event bus are analyzed effectively and precisely by applying specific rules and patterns through the event processing engine based on the CEP model that enables the multiple situation detections presented in this paper. This is discussed in detail in Chapter 4.

- Response and Action: The CEP model presented in this paper can be applied to complex business environments, and it can proactively respond to changes by automating the provision of useful business information and detecting signs of abnormality. It also enables administrators to take appropriate actions, such as acting on specific functions of enterprise systems or executing specific business processes of BPMS, in response to detected notable situations.

4. Design of an extended CEP Model

This chapter describes and details the complex event model for service monitoring considering the event processing architecture in the enterprise system proposed in Chapter 3. First, the design criteria are presented to identify differences from existing studies and to lay the groundwork for defining an extended CEP model. Second, events and the event meta-model are formally defined, providing a solid foundation for event handling. Third, we propose that the syntax and semantics of the constructs are written using an event language, including patterns, event operators, rules. Fourth, we present an event context, which is one of the key

components of the CEP model, to detect more contextual information from a collection of collected events.

4.1 Event definitions

An event can be defined as an instance that occurs at a specific point in time and can have significant meaning in the domain of interest [3]. An event can be represented as an event instance that contains the necessary information, such as the time and location of the event. The event type is described at the abstraction level by extracting common attributes of event instances.

In this paper, we define the event type as "E" or "e" and formally define it as follows.

Definition 1:

$E = (id, a, c)$.

Each event is identified by a unique id, where "a" is the set of attributes that determine the characteristics of the event, $a = attr_1, attr_2, \ldots, attr_n$, and $n \geq 0$. The causality vector "c" [15] causes the event that occurred, encompassing causally related events, $c = e_1, e_2, \ldots, e_n$, and $n \geq 0$. Causality vectors facilitate causal analysis between events by deriving cause-effect relationships between events using causal operators. The event type is defined using the formal specification technique as shown in **Fig. 2**.

The set of event types ΣE is a finite set $\Sigma E = \{E1, E2, ..., En\}$, $n \geq 0$. An event type E is a tuple $E = (id, a, c)$ where id is a unique identifier (event name) such that $\forall Ei, Ej \in \Sigma E, i \neq j : Ei.id \neq Ej.id.$, $a = \{attr_1, attr_2, ..., attr_n\}$, $n \geq 0$ is a finite set of attributes, and $c = \{e_1, e_2, \ldots, e_n\}$, $n \geq 0$ is a finite set of causality vector. An attribute $attr$ is a tuple $attr = (id, type)$ where id is a unique identifier (attribute name) such that $\forall E \in \Sigma E, \forall attr_i, attr_j \in E.attrs, i \neq j : attr_i.id \neq attr_j.id$ *and type is attribute type* $\in \{number, boolean, string, dateTime\}$.

Fig. 2. Formal specification of the event type

4.2 Event meta-model

The event meta-model is a model that defines the format required to receive events from the event sources such as services, activities, and databases to analyze the collected event data and provide useful information at the business level [16]. Events are not independent of each other, but are strongly related and can be represented using a formal meta-model, as shown in **Fig. 3**.

The proposed event meta-model is defined based on the OASIS web services distributed management event format (WEF) [16] standard and can be extended and used in WEF-compliant event processing systems and service monitoring tools, providing a basis for an effective event processing and analysis. Events are categorized into composite and primitive events, both of which are characterized by event properties, and there is a causal relationship between the events. Operators combine events to form complex events or situations and are classified into logical, time, or causal operators. Event context is needed for a more delicate multi-state detection while transforming the collected low-level to high-level events to provide useful business information, and it includes semantic space and an abstraction hierarchy. Details about event context are discussed in section 4.5, respectively.

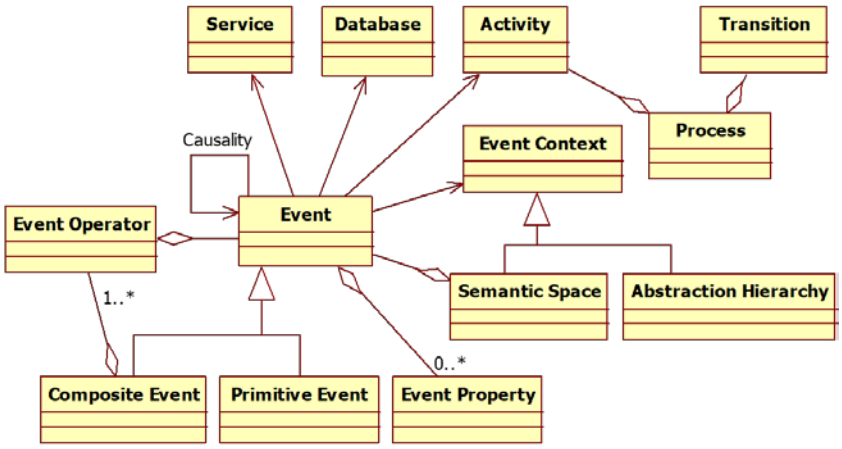

Fig. 3. Event meta-model

4.3 Event processing rules

Definition 2: Complex events can be specified as follows.

Complex Event name [subof Event Names]{

Pattern {EACH} operator (operand {(CONDITION)}, …)

[Where { [equivalence test], parameterized predicates } {CONTEXT context}

{WITHIN, INTERVAL, AT}]

[Action action]

}

After the “Pattern” keyword, the event pattern is specified to begin detection. In the condition clause after the “Where” keyword, the condition is also specified to trigger this complex event. After the “Action” keyword, the name of the method for event handling is specified. The conditional and event handling clauses can be omitted.

The optional items are grouped in braces “{}.” The “EACH” keyword means that all instances of a complex event should be reported. If you do not specify “EACH,” only the first composite event instance is reported. The term “CONDITION” is used to search a specific event instance for example, (reader = ”05AE”). The equivalence check in the conditional clause checks whether the values for the same attributes are identical between events, and the parameter condition specifies the constraints between the operands as parameters.

The “CONTEXT” keyword is followed by context information to detect multiple more delicate situations. The “WITHIN,” “INTERVAL,” and “AT” keywords indicate the time range, interval time, and specific point time of each complex event, respectively.

4.4 Definition of event operators

Complex events differ in their expressiveness depending on the event operators provided in the event model [17]. The event operator proposed in this study is designed to be suitable for

service monitoring of enterprise systems by improving and extending existing studies.

- Hierarchy support operator: One of the key concepts for dealing with complex events is the hierarchical structure of events. The event layer support operations provided in this paper are as follows. When defining an event, use the “subof” operator to specify the event of the upper layer. In the specification of a complex event, the event's name automatically reacts to all events in the lower layers, including itself. If you want to exclude child events, you can use the operator and follow the event name with “!”. For example, if the complex event “CChildEvent” is declared as a sublayer of “CParentEvent” (that is, CChild-Event subof CParentEvent), then the “CParentEvent” event contains a “CChildEvent,” but the “CParentEvent!” event will not contain a “CChildEvent.” For example, the “EventA–EventC! or EventB” complex event reacts to sub events containing “EventA” or sub events containing “EventB” but not to “EventC.” When “EventC” is a sub-event of EventA, **Fig. 4** shows that only some of the independent event layers can be selected using the event layer operation.

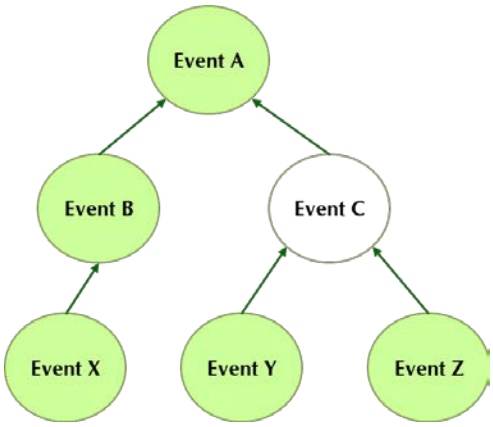

Fig. 4. Event hierarchy structure

- Event instance reference operator: In this paper, we provide an extended model based on existing studies by providing an instance specification operation that can refer to a specific event instance. When referring to a specific event instance, the event instance you are creating can be referred to as “this,” and you can point to the previous instance of “k” by using “prev(k).” If used without parameters (ie prev), it is the same as “prev(1)”. In front of an event instance, you can use the “@” operator to obtain the timestamp of the event instance. For example, “@prev(1)” shows the timestamp of the last instance of the same complex event. The “R(Relative)” operator allows you to pick the most recent instance of events that have occurred up to that point in time. You can also append the “@” operator to obtain the timestamp of the instance. **Fig. 5** shows the event occurrence history of two types of events in a system. For example, event e_1 was detected at system times 1, 2, 3, 5, and 6, and an event instance is generated for each occurrence. Using the previous notation, @(e_1, 1) = 1, @(e_1, 2) = 2, @(e_1, 3) = 3, @(e_1, 4) = 5, and @(e_1, 5) = 6. R(e_2, @e_1, 3) points to an

instance of type e_3 before the third, based on the timestamp of the most recent instance of e_1. Thus @R (e_2, @e_1, 3) = 2.

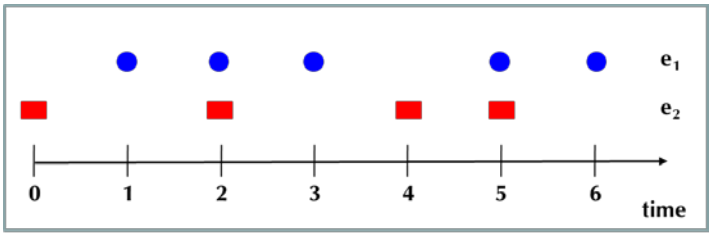

Fig. 5. Event Instance creation

- Time constraint operator: Existing studies only consider operations at the specific time when an event occurred. In this study, the operator is extended so that the time interval can be specified. The event time can be classified into point time and interval time. In **Fig. 6**, the time t_2 when the occurrence of the event e_1 is completed is the designated time of e_1, and [t_1, t_2] is the interval time of e_1. The definition is as follows. When T(e) indicates the time of event e, T_b(e) indicates the start time and T_e(e) indicates the ending time. In this study, it is assumed that there is a strong temporal relationship when interval times of different events are mutually independent of each other and that there is a weak temporal relationship when interval times of different events overlap. This is defined as follows.

Definition 3: Strong time relationship.

$$T(e_1) < T(e_2) \Leftrightarrow T_e(e_1) < T_b(e_2)$$

Definition 4: Weak time relationship.

$$T(e_1) \prec T(e_2) \Leftrightarrow T_b(e_1) < T_b(e_2) \wedge \{(T_e(e_1) > T_e(e_2)) \vee (T_e(e_1) < T_e(e_2))\}$$

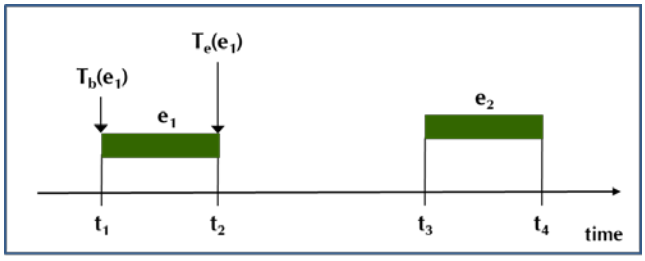

Fig. 6. Point time and Interval time

4.5 Event context

The event context mechanism is used to define additional information needed to transform the low-level to high-level information for a more delicate multi-situation detection. An event context includes elements such as abstraction hierarchy and semantic space for different dimensions (e.g., product and time). The semantic space is a temporary context related to

detect situations. The Temporariness is because the semantic space is a time window containing several situations. The semantic space contains context information that is relatively independent of the event data, such as location, role, and state, as shown in **Fig. 7**. The semantic space triggers two events, initiator and terminator events, to establish the boundary between beginning and end. In the semantic space definition, if the conditions defined by "condition" of the initiator and terminator are met, the semantic space is created and terminated respectively.

The abstraction hierarchy defines a range of attributes defined in the semantic space in the form of a hierarchical structure. The defined attributes may be objects that apply to different specifications such as location, organization, and product according to the nature of the complex event. For example, in **Fig. 7**, the item "Shop Floor," defined under semantic space, can be layered from higher to lower levels for the location. By using the abstraction layer as described above, it is possible to abstract event properties while processing various events, thereby making it easier to classify the events. In other words, abstraction enables delicate event analysis by detecting events that are the root cause of the interest and the source of the lowest layer of the event.

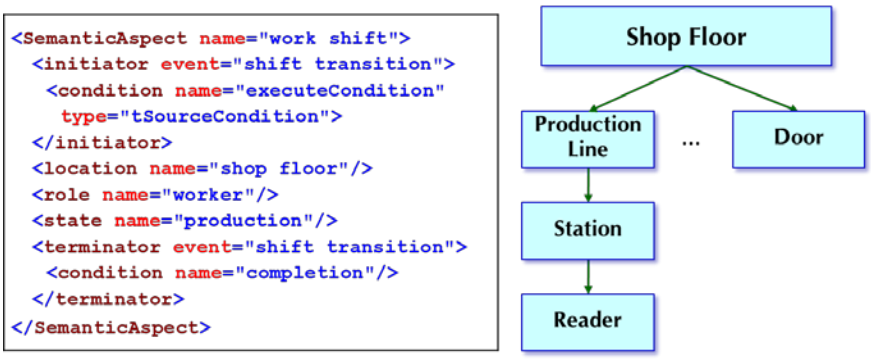

Fig. 7. Example of semantic space and abstraction hierarchy

The proposed CEP model supports the specification of formalized event types and instances and provides various rich operators, rules, and functions to apply to and express complex business environments, as opposed to the existing simple ECA-based event specification and processing. The proposed model is designed for ease of use and provides an event meta-model, event context mechanisms for more delicate event analysis.

5. Case Study

In this section, a case study applies two scenarios, refrigerator manufacturing production and internet e-commerce, to show the applicability of the proposed CEP model. The scenario is first described, the notable situations that can occur in the scenario are then specified using the proposed model, and then various event types having discrete distributions for the corresponding events are defined and evaluated.

5.1 Applications

Scenario 1

In the workflow to be applied, an RFID reader is installed at the main positions of each production line, collecting events through the manufacturing task of a refrigerator production factory of a consumer electronics company. **Fig. 8** shows the business process of a composite service. The Apache ODE [18], which is an open source implementation that complies with the OASIS Web Services Business Process Execution Language (WS-BPEL) [5] 2.0 standard, is used for business process modeling. (1) to (6) represent the parts of the business process model (activity) in which the sequence of the workflow is mapped. (1) The refrigerator body and pre-applied packaging linings are prepared at the same time, and when both are ready, the process begins. (2) The worker assembles the body such as the bottom plate and the back plate of the refrigerator, and (3) the foam molding operation to produce the product by dispersing bubbles in the polymer resin and (4) inputting the corresponding compressor for each model. Assemble your refrigerator and compressor. (5) Check the items such as size, shape, and distribution of the cells in the foaming process and if the quality is not appropriate, the refrigerator returns to step (3). (6) Finally, use the prepared packing lining to pack.

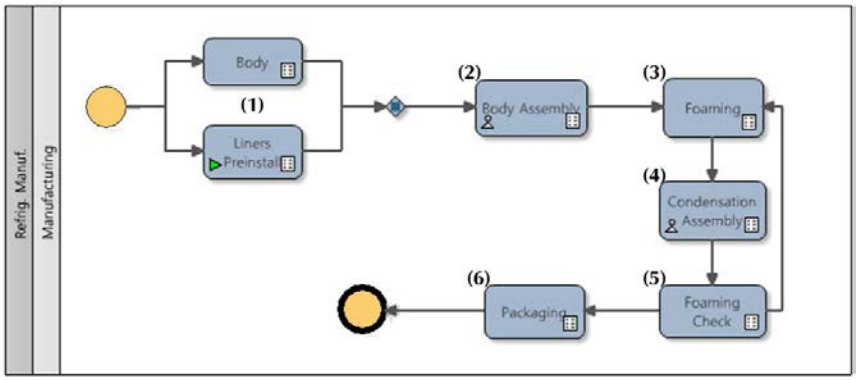

Fig. 8. Business process model of refrigerator production process

The proposed model is applied to two aspects of operation and management for the business process defined above. First, it is applied to an operation, i.e., for the erroneous assembly between the parts and the refrigerator in the assembly process. For example, in the following, when assembling a refrigerator and a compressor in the “Condensation Assembly” activity, the types of the refrigerator and compressor are identified using a sensor, such as an RFID reader. If an incorrect compressor is connected and the event interval time is 10 minutes, a complex event is detected to inform the worker that the wrong type of compressor is being assembled. Event context information called “work shift,” specified after the CONTEXT keyword in the conditional clause, is stored in the semantic space through the location, role, state information, and location information defined by the hierarchical structure of the abstraction layer. It is generated during the production process from the reader that was used. That is, it is possible to pinpoint the place where the root cause is located. The event processing language for this purpose is defined as follows.

```
Complex TypeCheck {
  Pattern (A=AND (REFRIGERATOR, OR
        (COMPRESSOR01, COMPRESSOR02)) )
```

```
Where ([compressor_type] and A == NULL
        CONTEXT work shift INTERVAL 10 min )}
```

The following is a concrete example of specifying a complex event by detecting complex events through the collection of events generated during the manufacturing process of the refrigerator production line through the MES and ERP systems. The model checks whether refrigerator-id corresponds to events x (reader input value = “05AE”) and y, which occur sequentially in/out of a specific location, and it detects a complex event that terminates in 10 minutes by referring to the parameter for the size of the weight and the event context information such as work shift.

```
Complex WeightCheck {
  Pattern ( EACH SEQ (IN_FOAM_ROOM (reader=”05AE”) x,
          OUT_FOAM_ROOM y)
  Where ( [refrigerator-id] and x.weight>y.weight CONTEXT work shift
          WITHIN 10 min )
  }
```

Second, it is applied to management: work time and quality control and delivery service management. The following are the complex events that help analyze the work time and quality of every person in every station according to the accurate records of primitive events WORKTIME and QUALITY.

```
Complex QualityCheck {
  Pattern ( EACH SEQ (WORKTIME, QUALITY) )
  Where ([person_id, station]
        INTERVAL  AUGUST)}
```

The following is an example of detecting a delivery error during a product's delivery service. The TRUCK event specifies that a truck, a means of delivery, is ready, and the EXIT-READING event is a complex event that is detected when the product or consumer information is incorrect after checking the contents of context.type.

```
Complex ShipmentCheck {
  Pattern ( EACH SEQ (TRUCK,
          EXIT-READING (type != context. type )) )
  }
```

Scenario 2

This is a scenario of internet e-commerce where a consumer finds a product in an internet-based e-commerce system, moves it to a shopping cart, and makes a payment. The following example detects an event where a consumer completes a purchase procedure in one

minute by selecting a product, moving the selection list to the shopping cart, and making a payment. If this pattern is repeated for the same item, it raises a FrequentSamePurchase event. If these complex events occur frequently in a short period of time, it can be suspected that the purchase system has been abnormally busy owing to a system failure.

```
Complex FrequentSamePurchase {
   Pattern ( SEQ(Find as x, MoveToCart as y, Pay, 1min) )
   Where ( [product_id] and @this - @prev <= 1min )
}
Complex FrequentSamePurchaseMoreThan5Within10min {
   Pattern ( COUNT(FrequentSamePurchase , ">=", 5, 10min) )
}
```

The following is an example in which an automatic recovery system detects a situation where an error occurs and generates an event in the link in the online payment process on the Internet e-commerce system. Among all events related to link error, the SystemAllShutDown event is excluded.

```
Complex AutoRepairNotOccurWarning {
  Pattern (SEQ( (LinkDown - SystemAllShutDown ) as
          x, (LinkDown - SystemAllShutDown) as y ) )
  Where ( [id] and @R(RepairTry, @y, 1) < @x)
}
```

In addition, the following event can be defined to add the reporting behavior to the administrator every 10 minutes when AutoRepairNotOccurWarning occurs.

```
Complex InfrequentAutoRepairNotOccurWarning {
  Pattern ( AutoRepairNotOccurWarning as x )
  Where ( @this - @prev >= 10min )
  Action Call( Message.SendAdminWarning(x) )
}
```

5.2 Evaluations

The prototype of monitoring framework is implemented by Eclipse and PostgreSQL, which name is SME (service monitoring in enterprise)-CEP. Two components have been implemented for ease of experimentation and for enhancing the usability of the monitoring framework. First, the event modeler component is used to easily comprehend event models which include elements such as operators, rules, patterns, and context; complex event types

are defined by combining them by using event configuration forms and by maintaining event models. Second, the schema transformer component makes it easy to convert the event definitions, specified in a complex event processing language, into XML, and to automatically generate them within the event model repository's database. This enables the ability to dynamically add, create, delete, and update the event models. **Fig. 9** shows this process.

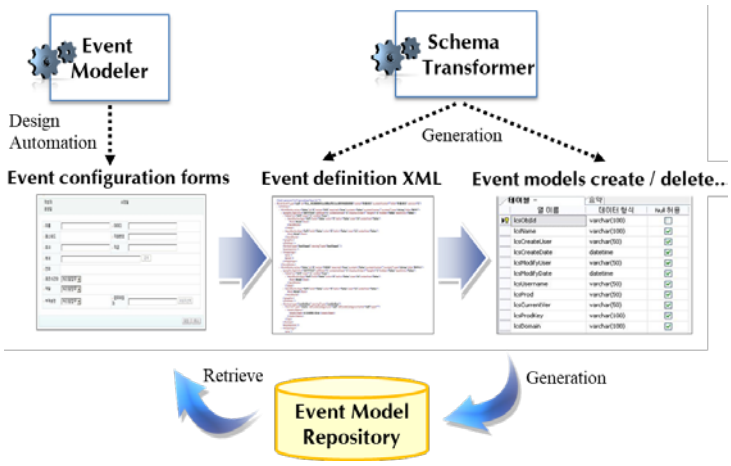

Fig. 9. Event model maintain process

Event configuration forms display a list of all the patterns, conditional expressions, and operators that can be retrieved from the event models; they also define complex events by combining the desired data, event operators, patterns, and so forth. **Fig. 10** defines a complex event called 'TypeCheck' by setting and combining database tables, patterns, conditional expressions, operators, and any other relevant elements. More detailed information on monitoring framework implementation is beyond the scope of this paper; the purpose of this chapter is to provide an understanding of the experimental and evaluation methods.

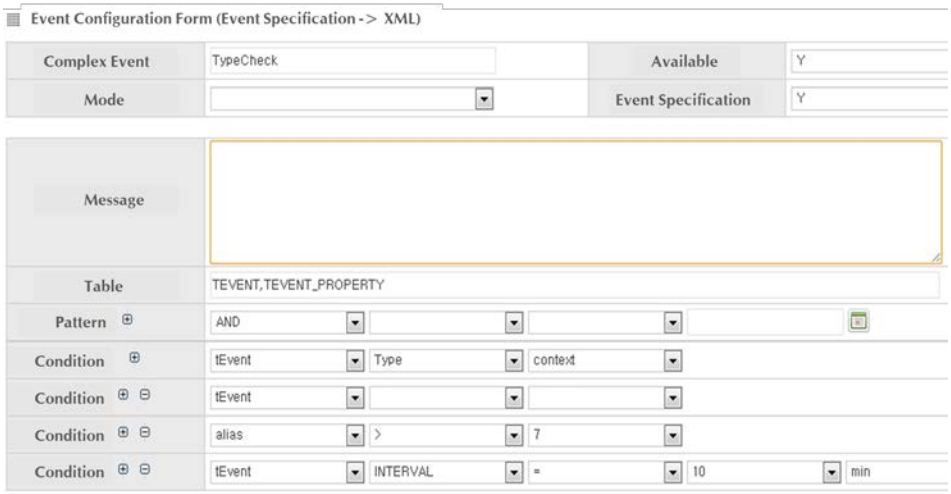

Fig. 10. Event configuration form

There are 14 event types for evaluations, namely {E_1, E_2,…,E_{14}}, each of which have simple attributes and methods to obtain their attributes. And there are also 50 complex event expressions. Each event type is used by at least one complex event expression that takes it as a component event. Such an environment for testing is set for application, as shown in section 5.1. The event instances are generated randomly. For each input event, 10 tests are conducted, and its mean value is used for the analysis. To date, there is no generally accepted solutions for run-time monitoring of services provided by an enterprise using CEP. The similar solutions are Amit [19], Esper [20], etc. Amit is embedded in other commercial software products, that is, they are neither independent products nor available. Esper is an independent product, which aims at addressing the requirements of applications that analyze and react to events. And it is open source software. Wang's Event Stream Suppression Model (ESSM) [11] has declarative language for defined complex events and supports event operators based on event instances. Thus, Esper and ESSM was used for performance comparison. **Table 1** shows the results of the comparative experiments applied to the refrigerator production process using the test techniques and experimental scenarios. Similar to the performance measurement experiment, the time required, and the number of detected situations were measured by generating 10 measurements per 100,000 events.

Table 1. Evaluation results (Refrigerator production scenario)

unit: number

Evaluation Item / Number of generated events	ESSM		Esper		SME-CEP	
	Trip Time (ms)	Detected situation	Trip Time (ms)	Detected situation	Trip Time (ms)	Detected situation
100,000	900	910	913	990	1,012	10,103
200,000	7,910	10,000	8,025	10,018	9,015	200,015
300,000	16,020	19,910	18,090	21,110	19,031	400,101
400,000	28,110	20,044	31,001	30,059	32,080	550,021
500,000	39,550	39,000	45,022	51,112	50,077	700,011
600,000	55,000	55,100	63,023	72,020	67,007	870,002
700,000	76,120	62,000	80,011	90,105	83,025	990,011
800,000	88,200	102,000	90,101	110,005	98,021	1,130,020
900,000	92,890	110,003	100,012	120,082	110,003	1,250,015
1,000,000	101,120	119,990	111,022	130,220	124,319	1,391,110

In the evaluations result, the occurrence event refers to the total number of event instances generated through the experiment scenario, and the detected situation is the total number of detected complex events during the process of receiving and processing the generated event instance. The analysis results show that the number of detected situations increases significantly with the increase in the number of event instances generated using Esper and ESSM compared with the proposed model in this study. This shows that the proposed method performs much better in detecting situations for decision making. This is possible because this

paper's proposed method provides a wider variety of operators, rules, and functions than the other two methods; it also enables more refined event analysis to be performed by providing mechanisms such as event context. With respect to performance, it is apparent that the proposed method requires additional processing time over current methods, because event history buffer operations are frequently triggered by the detection of a greater variety of situations than before. Calculation of throughput using measurements of the time required reveals that this method detects about 11,189 notable situations per second, that Esper detects about 1,172 per second, and that ESSM detects about 1,186 per second. **Fig. 11** shows the trip time and number of situations (complex events) detected using the three methods.

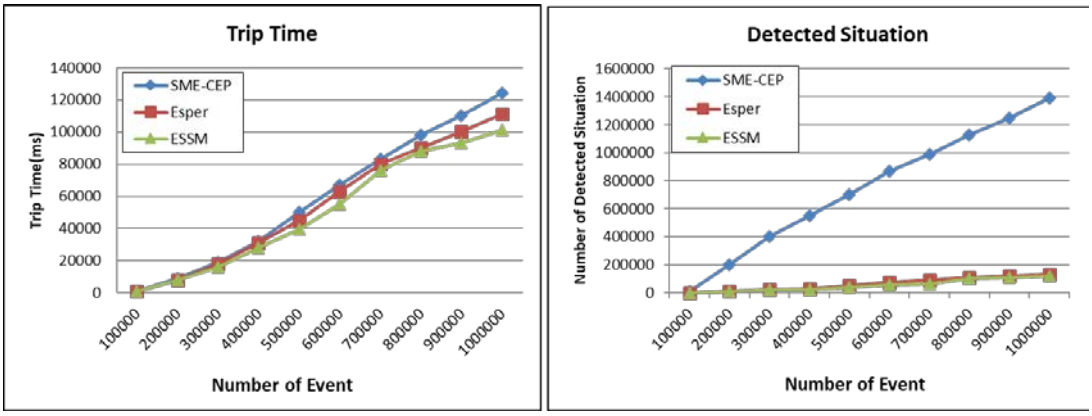

Fig. 11. Evaluations results (Refrigerator production scenario)

Table 2 shows the results of a comparative experiment applied to the Internet e-commerce scenario using the same test technique and experimental scenario.

Table 2. Evaluation results (Internet e-commerce scenario)

unit: number

Evaluation Item / Number of generated events	ESSM		Esper		SME-CEP	
	Trip Time (ms)	Detected situation	Trip Time (ms)	Detected situation	Trip Time (ms)	Detected situation
100,000	890	705	909	908	1,003	10,010
200,000	15,400	1,980	17,078	10,020	19,007	210,503
300,000	27,690	18,700	31,009	30,717	34,101	420,008
400,000	48,900	29,010	51,720	45,250	54,307	560,077
500,000	59,020	39,000	63,991	60,202	70,011	690,001
600,000	67,980	45,000	79,048	70,009	85,033	800,303
700,000	79,688	52,300	89,120	80,110	96,021	930,712
800,000	90,033	61,000	99,031	90,012	110,030	1,030,015
900,000	99,344	68,700	110,079	99,079	119,500	1,150,520
1,000,000	100,769	83,400	119,260	110,910	132,345	1,282,600

When analyzing the results of **Table 2**, the service monitoring model method proposed in this paper performs better in detecting notable situations for decision making than does Esper and ESSM. By calculating the throughput using the time required, this method detects about 9,691 notable situations per second and Esper detects about 929 and ESSM detects about 827. This study shows an improved performance in complex event processing. This demonstrates the merits of this study in terms of the proposed model's reliability and functionality. **Fig. 12** shows the graph of **Table 2**.

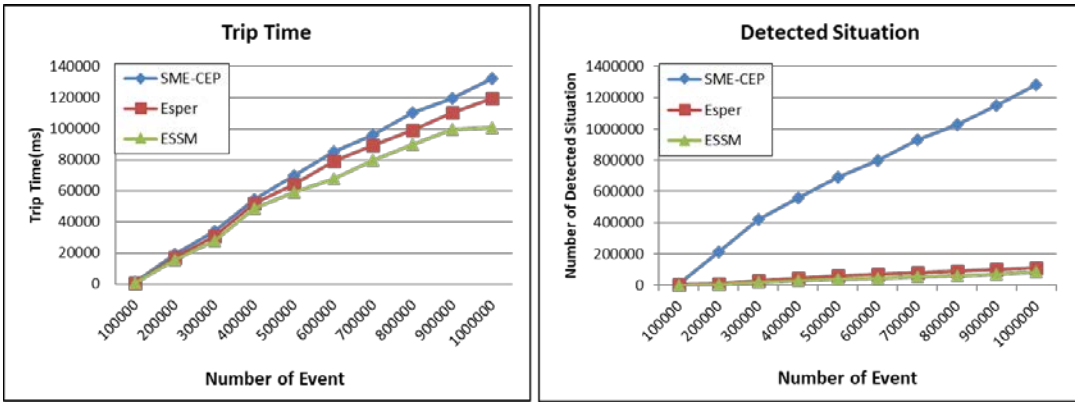

Fig. 12. Evaluation results (Internet e-commerce scenario)

6. Conclusion

In this study, an extended CEP model is proposed to provide more valuable and useful business information and early response through situation detection by effectively handling various events occurring in a complex business environment. The way in which the proposed approach can be implemented in enterprise systems is put forth. The meta-model, context, rules, key, and operators of event processing are formulated in detail. The results of this study are summarized as follows.

First, we proposed an enterprise event processing architecture to more efficiently collect, analyze, and respond to various events occurring in an enterprise environment, and we provided a basis for designing CEP models, especially an event meta-model.

Second, defining the event type and an event meta-model that clearly specifies the syntax and rules for event handling and provides a solid foundation for all the elements that make up the proposed event model. The proposed event meta-model is defined based on the OASIS WEF [16] standard and can be extended to WEF-compliant event processing systems and service monitoring tools.

Third, by supporting the hierarchical structure of events, the correlation between the events of the equal layer and the upper and/or lower layers is defined to simplify the complexity and facilitate the application and expression of the business environment.

Fourth, by providing an instance reference operator to specify not only the event type but also each event instance, and the operator to specify interval times in addition to specifying the exact timing constraints between event instances that occur. By doing this, the expression power of the model for the application of a complex business environment is increased.

Fifth, by providing an event context and key mechanism that makes it possible to detect more contextual information independently of event data and enables more delicate event analysis.

The application and evaluation results show that the proposed approach is effective in increasing the agility and responsiveness of enterprises. In the future, the rules of complex event pattern transformation from the business process model will be formulated and functionality automatically changing the related event processing rules, by analyzing the properties of the events generated during service execution, will be implemented.

References

[1] H. N. Sad and T. Noria, "A Novel Approach for Integrating Security in Business Rules Modeling Using Agents and an Encryption Algorithm," *Journal of Information Processing Systems*, vol. 12, no. 4, pp. 688-710, 2016. Article (CrossRef Link).
[2] T. Wang, S. Truptil and F. Benaben, "An automatic model-to-model mapping and transformation methodology to serve model-based systems engineering," *Information Systems and e-Business Management*, vol. 15, pp. 323-376, 2017. Article (CrossRef Link).
[3] K. M. Chandy, S. Ramo and W. R. Schulte, "What is Event Driven Architecture (EDA) and Why Does it Matter?," *Gartner Inc*., March 2007.
[4] K. M. Chandy, "Event-Driven Applications: Costs, Benefits and Design Approaches," *California Institute of Technology*, November 2006.
[5] OASIS Standard, Web Services Business Process Execution Language Version 2.0. OASIS, 11 April 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[6] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley, 2002.
[7] K. Lin, N. Panahi, Y. Zhang and S. Chang, "Building Accountability Middleware to Support Dependable SOA," *IEEE Internet Computing*, vol. 13, no. 12, pp. 16-25, 2009. Article (CrossRef Link).
[8] L. Baresi, S. Guinea, M. Pistore and M. Trainotti, "Dynamo + Astro: An Integrated Approach for BPEL Monitoring," in *Proc. of IEEE International Conference on Web Services (ICWS 2009)*, pp.230-237, 2009. Article (CrossRef Link).
[9] D. Luckham and B. Frasca, "Complex Event Processing in Distributed System," *Stanford University Tech, Report CSL-TR-98-754*, Mar, 1998.
[10] C. McGregor and J. Schiefer, "A Web-Service based framework for analyzing and measuring business performance," *Information Systems and e-Business Management*, vol. 2, no. 1, pp. 89-110, Springer, 2004. Article (CrossRef Link).
[11] D. Wang et al., "Utility-maximizing event stream suppression," in *Proc. of 2013 ACM SIGMOD International Conference on Management of Data*, pp. 589-600, June 2013. Article (CrossRef Link).
[12] A. K. Mok and G. Liu, "Efficient Runtime Monitoring of Timing Constraints," in *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 1997.
[13] M. Hellinger and S. Fingerhut, "Business Activity Monitoring: EAI Meets Data Warehousing," *eAI JOURNAL*, pp. 18-21, July 2002.
[14] M. Thirumaran and G. G. Brendha, "Incremental stages of a semantic framework for automating the changes on long term composed services," *Human-centric Computing and Information Sciences*, vol. 6, no. 1, pp. 1-26, 2016. Article (CrossRef Link).

[15] C. G. Lee et al., "Monitoring of Timing Constraints with Confidence Threshold Requirements," *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 977-991, 2007. Article (CrossRef Link).
[16] OASIS, Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1 and Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 2, 01 August 2006. http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf.
[17] G. Liu et al., "Composite Events for Network Event Correlation," in *Proc. of the IFIP/IEEE Symposium on Integrated Network Management*, 1999.
[18] Apache ODE. http://ode.apache.org/index.html.
[19] A. Asaf and E. Opher, "Amit-the situation manager," *The International Journal on Very Large Data Bases*, vol. 13, no. 2, pp. 177-203, May 2004.
Article (CrossRef Link).
[20] Esper. http://www.espertech.com/.

Deuk Kyu Kum is currently a professor in Dept. of Information and Communication Engineering, Yuhan University, Bucheon, Korea. He received the M.S. and Ph.D. degrees in Computer Science and Engineering from Soongsil University, Korea, in 2005, 2012, respectively. His research interests include big data analysis technology, internet and mobile computing, and cloud computing.