Journal of The Korea Society of Computer and Information
Vol. 22 No. 10, pp. 109-119, October 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.10.109

The Design of an Extended Complex Event Model based on Event Correlation using Aspect Oriented Programming

Deuk-Kyu Kum*

Abstract

In recent through development of IOT owing to that mass stream data is being generated in variety of application complex event processing technology is being watched with keen interest as a technology to analyze this kind of real-time continuous data. However, the existing study related with complex event processing only comes to an end at simple event processing based on low-level event or comes to an end at service defect discovery with providing limited operator and so on. Accordingly, there would be limitation to provide useful analysis information. In this paper in consideration of complex event along with aspect-oriented programming an extended complex event model is provided, which is possible to provide more valuable and useful information. Specifically, we extend the model to support hierarchical event structures and let the model recognize point-cuts of aspect-oriented programming as events. We provide the event operators designed to specify the events on instances and handle temporal relations of the instances. It is presented that syntax and semantics of constructs in our event processing language including various and progressive event operators, complex event pattern, etc. In addition, an event context mechanism is proposed to analyze more delicate events. Finally, through application studies application possibility of this study would be shown and merits of this event model would be present through comparison with other event model.

▸Keyword: Complex Event, Event Processing, AOP, Event Specification, Event Context

I. Introduction

최근 사물 인터넷과 센서 기술의 발달로 다양한 응용에서 대용량의 스트림 데이터가 생성되고 있다. 이와 함께, 빠르게 변화하고 있는 기업 환경에서 실시간 의사결정의 중요성이 증대됨으로써 의미 있는 정보를 제공하고 위험 또는 기회 발생 등의 변화에 신속하게 대응하도록 하는 것이 기업의 핵심 경쟁력이 되었다[1]. 이러한 상황에서 기업 내외에서 발생하는 이벤트를 실시간으로 분석하고, 처리함으로써 의사 결정 지원 및 조기 대응을 가능하게 하는 기술로 복합 이벤트 처리(Complex Event Processing)가 주목을 받고 있다[2, 3].

복합 이벤트 처리는 여러 이벤트 소스로부터 발생한 이벤트를 대상으로 이벤트들의 영향을 분석하여 의미 있는 정보를 제공하고, 대응되는 액션을 처리하는 기술로 가장 핵심이 되는 복합 이벤트[4]는 이벤트들 간의 연관 관계를 이용해서 정의되며, 이벤트-조건-행동(event-condition-action, ECA)규칙을 이용한 체계적인 구현 방법을 제공할 수 있다. 하지만, 기존 복합 이벤트 처리 연구는 낮은 단계의 이벤트에 기반한 단순 이벤트 처리에 그치고 있거나, 제한된 이벤트 연산자 제공 등으로 서비스 결함 발견에 그치고 있어 보다 의미 있고 유용한 분석 정보를 제공하기에는 한계가 있다.

본 논문에서는 복합 이벤트와 관점지향 프로그래밍(Aspect Oriented Programming: AOP)을 함께 고려하여 다중 상황 검출(situation detection)을 통해 보다 가치 있고 유용한 비즈니

• First Author: Deuk-Kyu Kum, Corresponding Author: Deuk-Kyu Kum
*Deuk-Kyu Kum (dkkum@hanmail.net), Dept. of Information Technology, Yuhan University
• Received: 2017. 09. 25, Revised: 2017. 10. 11, Accepted: 2017. 10. 21.

스 정보의 제공을 가능하게 하는 확장된 복합 이벤트 모델을 제시하고, 제안된 이벤트 모델이 어떻게 응용될 수 있을지 보인다. 구체적으로 본 연구에서는 AOP의 교차점을 일반적인 복합 이벤트로 인식하도록 하고, 클래스나 관점에 독립적인 복합 이벤트 계층을 정의할 수 있도록 복합 이벤트의 정의 방법을 확장한다. 또한, 이벤트 타입뿐만 아니라 인스턴스 단위로도 이벤트 명세를 할 수 있도록 연산자를 제공하고, 이벤트 인스턴스 간에 시간적 관계를 원활하게 표현할 수 있도록 한다. 이러한 다양하고 진보된 이벤트 연산자와 복합 이벤트 패턴, 규칙 등 이벤트 처리 언어를 구성하는 요소의 문법과 의미를 제안한다. 또한, 보다 정교한 이벤트 분석을 위한 이벤트 컨텍스트 매커니즘을 제안한다. 마지막으로 응용사례를 통하여 본 연구의 효과성과 적용 가능성을 보여주고, 다른 이벤트 모델과의 비교를 통해 본 이벤트 모델의 장점을 제시한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 복합 이벤트 처리 기술과 AOP를 소개하고 관련연구를 서술하며, 3장에서는 복합 이벤트 모델의 설계 기준과 이를 준수하는 이벤트 모델을 설계하고 각 요소 별로 자세히 설명한다. 5장에서는 응용 사례를 기술하고, 6장은 기존 연구들과의 비교를 통해 제안된 이벤트 모델의 우수성을 보인 후, 마지막 7장은 논문을 요약하며 향후 연구를 제시한다.

II. Preliminaries

1. Related works

1.1 CEP (Complex Event Processing)

복합 이벤트 처리 기술은 여러 이벤트 소스로부터 발생한 이벤트를 대상으로 실시간으로 의미 있는 데이터를 추출하여 대응되는 액션을 수행하는 것을 말한다[3]. 이때 이벤트 데이터는 스트림 데이터로써 복합 이벤트 처리의 핵심 기술은 이러한 데이터를 실시간으로 분석하여 주어진 시간 제약 조건 안에서 의미 있는 이벤트와 그들의 패턴들을 식별하는 것이다 [4, 5]. 이벤트를 다루기 위한 기술은 이벤트 처리 방법에 따라 크게 다음과 같이 나눌 수 있다[5, 6].

- 단순 이벤트 처리: 발생한 이벤트들은 모두 의미 있는 이벤트로 간주하고 각각의 이벤트 내용에 따라 대응되는 액션을 수행한다. 발행/구독 방식이나 중재 방식에 의해 이벤트 처리를 제공한다.
- 스트림 이벤트 처리: 의미 있는 이벤트와 무의미한 이벤트가 같이 발생하는 대량의 이벤트 스트림을 대상으로 하며, 필터링 등을 수행하여 의미 있는 이벤트 정보만 뽑아서 응용에 전달 혹은 서비스와 연동한다.
- 복합 이벤트 처리: 여러 이벤트 소스로부터 발생한 이벤트를 대상으로 이들 이벤트들의 조합의 의미를 분석하여 유용한 정보를 제공하는 것을 의미한다[4, 5]. 단순 이벤트 처리가 하나의 이벤트를 대상으로 한 반면, 복합 이벤트 처리는 여러 이벤트간의 다양한 관계를 분석한다.

복합 이벤트 처리의 프로세스는 그림 1과 같이 다양한 센서 데이터와 시스템간의 상호작용 등을 중요한 이벤트로서 식별하고, 미리 정의한 패턴들과 이 이벤트들을 관련시켜 이들 조합의 의미를 분석한 후 의사 결정에 중요한 정보나 상황을 검출하여 이에 대한 대응 액션(action)을 취하는 데에 기본적인 바탕을 제공해 준다.

Fig. 1. Complex Event Processing Process

1.2 AOP (Aspect Oriented Programming)

AOP는 1997년 Gregor Kiczales 등이 참여하여 제안한 프로그래밍 개발 방법론이다[7]. 프로그래밍 개발 방법론은 일괄처리 프로그래밍(Batch Process Programming)에서 구조적 프로그래밍(Structured Programming), 객체지향 프로그래밍(Object Oriented Programming)으로 발전해왔다. 구조적 프로그래밍과 객체지향 프로그래밍은 함수를 이용하여 요구사항을 모듈화할 수 있는 방법을 제공한다. 개발자는 어떤 시스템에서 다루고자 하는 도메인을 관심사(concern)로 구분할 수 있는데, AOP에서는 관심사를 핵심 관심사(Core concern)와 횡단 관심사(Cross-cutting concern)로 분류한다. 그리고 구조적 프로그래밍이나 객체지향 프로그래밍과 같은 기존의 프로그래밍 방법론으로 모듈화하기 어려운 요구 사항을 횡단 관심사로 분류하고, 이를 관점(Aspect) 단위로 모듈화하여 모듈의 재사용성을 높였다.

AOP에서는 각 관심사가 독립적으로 구현되며, 다른 관심사와는 최대한 느슨하게 결합될 수 있는 방법을 제공한다. 따라서 관점지향 프로그래밍은 각 모듈에 대한 명확한 책임 소재와 시스템 개선의 용이, 코드 재사용의 확대 등의 장점을 갖는다. 관점은 교차점과 충고(advice)로 이루어진다. 교차점은 하나 이상의 결합점들을 술어(predicate)를 이용하여 묶은 것인데, 결합점은 프로그램 수행 중 가로챌(intercept) 수 있는 지점에 해당하며 AOP의 구현에 따라 그 미세함(granularity)이 달라질 수 있다. 교차점을 이루는 결합점은 대개 메쏘드 이름을 이용한 패턴 등으로 정의된다. 프로그램의 수행 중 해당 패턴이 나타나면 적용 시점(before, after, around)에 따라 충고가 수행된다.

본 연구에서는 AOP의 교차점을 일반적인 복합 이벤트로 인식하도록 하고, 클래스나 관점에 독립적인 복합 이벤트 계층을 정의할 수 있도록 복합 이벤트의 정의 방법을 확장한다. 또한, 관점을 보다 정교한 이벤트 분석을 위해 제안된 이벤트 컨텍스트 매커니즘의 의미 관점(Semantic Aspect)으로 응용하여 다중 상황 검출을 가능하도록 한다.

1.3 McGregor's Solution Manager Service[8]

McGregor의 연구에서는 비지니스 프로세스 성능 측정을 위한 솔루션 매니저 서비스 (Solution Manager Service, SMS) 아키텍처를 제안하였다. 제안된 아키텍처는 웹서비스 로그를 지원하며 이벤트 프로세싱 컨테이너 (Event Processing Container)를 통하여 실시간에 많은 수의 프로세스 이벤트들을 처리할 수 있도록 하였다. 또한, 제안된 솔루션은 정의된 규칙을 적용한 이벤트 처리 결과에 대한 대응을 자동화 할 수 있도록 하였다. 하지만 이벤트 처리 대상이 낮은 단계의 이벤트(low-level event)에 기반한 단순 이벤트 처리에 그치고 있어, 기업에 유용한 비즈니스 정보를 제공하기에는 한계가 있으며, 시간 및 인과 관계에 대한 연산자는 제공하지 않았다.

1.4 Luckham's RAPIDE[9]

Luckham의 연구에서는 복합 이벤트 처리의 구현을 위해 RAPIDE 언어를 개발하였으며 이벤트의 연관성(causality) 정보를 사용하여 이벤트 poset(partially ordered set of events)를 정의하였다. 이를 바탕으로 이벤트 패턴, 추출, 결합(event pattern, filtering, aggregation) 등의 기술을 사용한 이벤트 모델을 제안하였다. 이벤트 명세를 표현하기 위한 언어로서 ECA 규칙을 사용하여 시스템에서 벌어지는 상황에 따라 자동적으로 반응하도록 설계되었다. 특징으로는 이벤트 수식어와 복합 이벤트 표현을 위한 이벤트 연산자를 제시하고 있으며, 이벤트 절은 복합 이벤트의 형식으로 표현되고 조건 절에는 부가적인 조건들을 기술한다. 하지만 수집되는 이벤트의 구체적인 형태는 언급하지 않았으며, 이벤트 인스턴스 단위의 명세도 고려되지 않았다.

1.5 Wang's Event Stream Suppression Model[10]

Wang은 이벤트의 타입에만 의존하는 복합 이벤트 정의 방법은 본질적으로 모호성을 가지고 있으며, 그에 대한 부분적 해결책인 이벤트 소비 모드(event consumption mode)도 유연하지 못하다고 지적하였다. 대안적인 복합 이벤트 정의 방법으로 실시간 시스템의 명세에 주로 사용되는 Real-Time Logic(RTL)[11]의 이벤트 모델을 확장하여 이벤트 인스턴스에 기반한 복합 이벤트 정의 방법과 관련 이벤트 연산자를 정의하였다. 제안된 모델은 이벤트 상관에 기반한 네트워크 관리 시스템 분야에서 연구 되었으며, 이벤트의 데이터 모델, 쿼리 모델, 패턴 모델의 3가지 카테고리를 대상으로 다양한 연산자들을 제공한다. 하지만, 이벤트 처리를 위하여 일반적인 ECA 규칙을 사용하였으며, 네트워크 관리 시스템이라는 특정 도메인에 초점을 맞추어 설계된 시스템으로 부분적인 솔루션만 제공함으로써 본격적인 복합 이벤트를 지원하여 정제된 상황 파악을 통해 원인 분석을 보다 쉽게 할 수 있도록 하기에는 한계가 있다.

III. The Design of an Extended Complex Event Model

1. Design criteria for complex event model

본 논문에서 제안하는 복합 이벤트 모델은 본격적인 복합 상황 인지를 통하여 정제된 상황 파악을 통해 효과적인 원인 분석과 조치를 취하는 것을 가능하게 하기 위하여 다음과 같은 설계기준을 적용하였다.

• **이벤트의 정형 명세:** 모호성 없는 이벤트 명세를 위해 이벤트 타입을 정형 명세 기법을 사용하여 정의하여 이벤트 메타모델 및 이벤트 처리를 위한 확실한 기초를 제공하고, 이벤트 타입뿐만 아니라 이벤트 인스턴스 단위로도 이벤트 명세를 할 수 있도록 인스턴스 지칭 연산자를 제공한다.

• **이벤트 계층 지원:** 이벤트를 계층적으로 구조화 시키는 것은 복잡성을 컨트롤하기 위한 디자인 기술이다. 기본(primitive) 이벤트의 경우에는 시스템 이벤트로서 미리 정의된 계층이 있는 경우가 많으나[12], 복합 이벤트의 계층 정의에 관한 연구는 활발하지 않았다. 본 연구에서는 복합 이벤트 계층에 대한 지원과 함께 계층 간의 관계를 나타낼 수 있는 연산자를 제공한다.

• **시간 및 인과 관계 연산자 제공:** 실시간 모니터링을 위해서는 발생되는 이벤트 인스턴스 간의 정확한 타이밍 제약에 관한 명세가 가능해야 한다[13]. 본 연구에서는 이벤트 인스턴스 간의 시간적 관계를 원활하게 표현할 수 있는 연산자를 제공한다. 특히, 지정 시간(point time)외에 간격 시간(interval time)을 지정할 수 있도록 연산자를 확장한다. 또한, 이벤트 간의 원인-결과 관계를 나타내는 직/간접 인과관계 연산자를 제공한다.

• **이벤트 컨텍스트 정보 제공:** 복합 이벤트 처리를 위한 다양한 연산자와 규칙, 기능을 제공하는 것 외에 보다 정교한 다중 상황 검출을 위해 이벤트 데이터와는 독립적으로 이벤트 컨텍스트 정보를 제공할 수 있는 매커니즘을 지원한다.

2. Event definitions

이벤트는 특정 시점에 발생하는 사건으로 관심 도메인 내에서 중요한 의미를 가질 수 있는 인스턴스로 정의할 수 있다. 이벤트는 필요한 정보 예를 들어 이벤트가 발생한 시간, 위치 등의 정보를 포함한 이벤트 인스턴스로 표현 될 수 있으며, 이벤트 타입은 이벤트 인스턴스들의 공통된 속성들을 추출하여 추상화 레벨에서 기술한 것이다.

본 논문에서는 이벤트 타입을 'E' 또는 'e'로 표기하기로 하고, 다음과 같이 정형적으로 정의한다.

정의 1: 이벤트 타입 E는 다음과 같이 정의될 수 있다.

$E = (id, a, c)$

각 이벤트 타입은 유일한 id로 식별되며, a는 이벤트의 특성을 결정짓는 애트리뷰트(속성)의 집합으로 a={attr1, attr2, …,attrn}, n≥0와 같이 정의한다. c는 발생한 해당 이벤트의 원인이 되는 즉, 인과관계가 있는 이벤트들을 포함하는 인과관계 벡터(causality vector)[14]으로 c={e1, e2, …,en}, n≥0와 같이 정의한다. 인과관계 벡터는 인과관계 연산자를 이용하여 이벤트 간의 원인-결과 관계를 도출하여 이벤트간의 행위 분석(behavior analysis)을 용이하게 한다. 이벤트 타입을 정형 명세 기법을 사용하여 정의하면 그림 2와 같다.

The set of event types ΣE is a finite set $\Sigma E = \{E_1, E_2, \cdots, E_n\}$, $n \geq 0$. An event type E is a tuple $E = (id, a, c)$ where id is a unique identifier (event name) such that $\forall E_i, E_j \in \Sigma E, i \neq j : E_i.id \neq E_j.id$. and $a = \{attr_1, attr_2, \cdots, attr_n\}$, $n \geq 0$, is a finite set of attributes. and $c = \{e_1, e_2, \ldots, e_n\}$, $n \geq 0$, is a finite set of event causality vector. An attribute $attr$ is a tuple $attr = (id, type)$ where id is a unique identifier (attribute name) such that $\forall E \in \Sigma E, \forall attr_i, attr_j \in E.attrs \in E.attrs, i \neq j : attr_i.id \neq attr_j.id$ and type is an attribute type, $type \in \{number, boolean, string, dateTime\}$.

Fig. 2. The Formal Specification for Event type

3. Complex event processing rules

원시 이벤트(raw event)는 이벤트 소스로부터 이벤트에 해당하는 상황은 검출되었으나 아직 이벤트 처리 시스템이 처리하도록 적합하게 기본 이벤트의 모습을 갖추지 못한 것을 말한다. 이러한 원시 이벤트는 흔히 동기화가 되지 않은 타임스탬프 혹은 중복된 데이터와 같은 노이즈가 포함되어 있을 수 있다. 다음은 기본 이벤트 명세(specification)를 위한 문법 및 구체적인 한 예를 보여준다. 제조라인의 설비운영 상황에서 발생하는 이벤트를 MES 시스템을 통하여 수집할 때의 예로 기본 이벤트 LightEvent는 LightSensor로부터 발생되며 세 개의 속성을 갖는다. 그 중 SerialNumber는 센서의 위치와 시간을 병합하여 만들고 나머지 속성은 해당되는 원시 이벤트의 속성을 가져온다. 노이즈 필터로써 LightFilter1 이후에 LightFilter2를 적용하고 있다.

정의 2: 기본 이벤트는 다음과 같이 명세할 수 있다.

Primitive 기본 이벤트 이름
From 원시 이벤트 소스
Attributes 속성 이름 정의, 원시 이벤트로부터의 대응
[NoiseFiltering 노이즈필터*]*

```
Primitive LightEvent
From LightSensor
Attributes {SerialNumber = Location + Time;
  LightIntensity = SignalStrength;
  Energy = RemainBattery}
NoiseFiltering {LightFilter1; LightFilter2}
```

정의 3: 복합 이벤트는 다음과 같이 명세할 수 있다.

Complex 복합 이벤트 이름 *[subof* 이벤트 이름들*]{*
Pattern 이벤트 패턴절 = *{EACH}* 연산자 (피 연산자 *{(CON)}, …)*
[Where 조건절 = *{[동치성 체크], 매개변수 조건}*
{CONTEXT 컨텍스트 정보*}*
{WITHIN, INTERVAL, AT}]
[Action 이벤트 처리절*]*
}

Complex 키워드 뒤에는 정의하고자 하는 복합 이벤트의 이름을 지정한다. Pattern 키워드 뒤에는 검출하고자 하는 이벤트 패턴을 지정한다. Where 키워드 뒤의 조건 절에는 부가적으로 이 복합 이벤트를 유발(trigger)시킬 조건을 지정한다. Action 키워드 뒤에는 이벤트 처리를 위한 메쏘드의 이름을 지정한다. 조건절과 이벤트 처리절은 생략 가능하다.

패턴절에서 중괄호 "{}"은 옵션 항목을 표시하고, EACH 키워드는 복합 이벤트의 모든 인스턴스가 보고되어야 하는 것을 의미한다. EACH를 명시하지 않을 경우 첫 번째 복합 이벤트 인스턴스만 보고된다. CON 키워드는 특정 이벤트 인스턴스를 검색하도록 사용된다. 예를 들어 (reader="05AE")와 같이 리더기를 통하여 입력된 값이나, 특정 이벤트의 아이디를 지정할 수 있다. 조건절에서 동치성 체크는 이벤트들 간의 동일한 속성(attributes)에 대한 값(value)이 동일한지의 여부를 검사하고, 매개변수 조건은 매개변수인 피 연산자간의 제약 조건을 명시한다. CONTEXT 키워드 뒤에는 보다 정교한 다중 상황을 검출하기 위한 컨텍스트 정보를 기술하고, WITHIN, INTERVAL, AT 키워드는 각각 복합 이벤트의 발생 시간 범위, 간격 시간, 특정 지정 시간을 표시한다.

다음은 복합 이벤트를 명세한 구체적인 한 예를 보여준다. 냉장고 생산 라인의 제조 공정 중 발생한 이벤트들을 수집하여 복합 이벤트를 검출하는 예로 특정 위치의 내/ 외부에서 순차적으로 발생한 이벤트 x(리더기 입력 값="05AE"), y에 대하여 refrigerator-id의 동일 여부를 체크하고, 매개변수인 중량의 크기 조건과 작업 교대라는 이벤트 컨텍스트 정보를 참조하여 10분 안에 상황이 종료되는 복합 이벤트를 검출한다.

```
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)
}
```

다음은 이벤트 패턴절에 나오는 이벤트 인스턴스를 as를 사용하여 임의의 변수로 표현할 수 있도록 하는 예이다. 이렇게 표현된 변수는 해당 이벤트 인스턴스를 대표하며, 해당 이벤트 인스턴스의 속성 등을 참고하기 위해 사용된다. 다음은 엔터프라이즈 시스템에서 제공하는 트랜잭션 서비스 중 발생하는 복합 이벤트를 검출하기 위한 예로 AbortTransaction 이벤트가 발생하면 그것의 속성인 critical의 값을 검사하여 참이면 Log.logmessage 메쏘드에 인자로 전달하여 호출한다.

```
Complex TransactionCancel {
  Pattern ( AbortTransaction as x )
  Where ( x.critical = true )
  Action Call( Log.logmessage(x) )
}
```

이어지는 연산자 정의에서 이벤트 연산자의 문법과 의미를 자세히 설명한다.

4. Defining Event Operators

복합 이벤트는 이벤트 모델에서 제공되는 이벤트 연산자에 따라 그 표현력(expressiveness)이 달라진다. 본 연구에서 제안하는 이벤트 연산자는 기존의 연구결과를 개선 및 확장하여 다중 상황 검출을 통해 보다 가치 있고 유용한 분석 정보의 제공 및 조기 대응을 가능하게 하는 이벤트 연산자를 설계한다.

• 계층 지원 연산자

복합 이벤트를 다루기 위한 주요 개념 중 하나는 이벤트의 계층적 구조화이다. 기본 이벤트의 경우에는 시스템 이벤트로서 미리 정의된 계층이 있는 경우가 많으나[14], 복합 이벤트의 계층 정의에 관한 연구는 활발하지 않았다. 본 연구에서는 AOP에서 정의된 교차점(named pointcut)들이 이벤트의 이름으로 인식된다. AOP의 애스펙트들 간에도 클래스처럼 상속이 가능하여 일견 애스펙트의 상속을 이용하여 이벤트 간의 그룹 관계를 나타낼 수 있는 것 처럼 보인다. 하지만, 이벤트 그룹정보는 일반적인 프로그래밍 언어에서의 상속과는 다른 성격의 것이다. 프로그래밍 언어의 상속은 소프트웨어 재사용을 위해 설계된 개념이며 구현 부분을 최소화한 인터페이스(interface)도 있지만 두 방법 모두 컴파일 단계에서 계층을 확정하기 때문에 유연하지 않은 접근 방법이다. 또한 각 응용마다 상이한 그룹 관계가 필요한 경우도 있기 때문에 본 논문에서는 응용 수준에서 복합 이벤트간의 그룹화를 효과적으로 제공해주는 독립적인 이벤트 계층 지원 방법을 제안한다.

본 논문에서 제공되는 이벤트 계층 지원 연산은 다음과 같다. 이벤트 정의 시 상위 계층의 이벤트 명시를 위해 subof 연산자를 둔다. 복합 이벤트의 명세 시에 이벤트의 이름은 자동적으로 해당 이벤트를 포함한 하위 계층의 모든 이벤트들에 반응한다. 만일 하위 이벤트들을 제외하고 싶다면 이벤트 이름 뒤에 ! 연산자를 쓰면 된다. 예를 들어 복합 이벤트 CChildEvent가 CParentEvent의 하위 계층으로 선언된 경우(즉, CChild-Event subof CParentEvent), CParentEvent 이벤트는 CChildEvent을 포함하지만 CParentEvent! 이벤트는 CChildEvent을 포함하지 않게 된다. 예를 들어 EventA-EventC! or EventB 복합 이벤트는 EventA를 포함한 하위 이벤트들 혹은 EventB를 포함한 하위 이벤트에 대해 반응하지만 EventC에 대해서는 반응하지 않는다. 특히 EventC가 EventA의 하위 이벤트일 때 다음의 그림 3은 독립적인 이벤트 계층 중 사용자가 원하는 일부만을 이벤트 계층 연산을 통해 선택할 수 있음을 보인다.

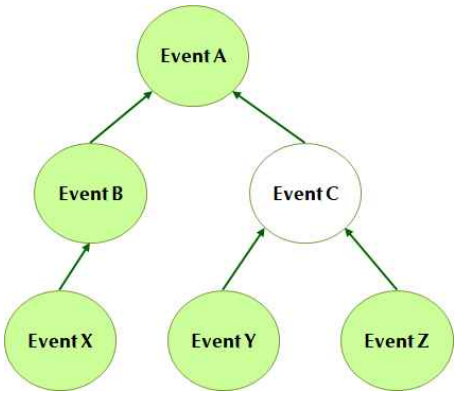

Fig. 3. Event hierarchy structure

• 이벤트 인스턴스 지칭 연산자

이벤트는 이벤트 타입에 의해 정의되며, 이벤트 타입은 많은 이벤트인스턴스를 가지고 있을 수 있다. 이벤트 타입으로만 명세에 표시하면, 복합 이벤트의 검출 시에 후보가 되는 이벤트의 인스턴스가 다수 존재할 때 어느 인스턴스를 사용하며 어떻게 이력 버퍼(history buffer)에서 삭제할 지에 대한 해결이 필요하다.

본 논문에서는 확장된 이벤트 소비 모드를 지원하며, 특정 이벤트 인스턴스를 지칭할 수 있는 인스턴스 지정 연산 역시 제공함으로써 기존 연구에서 확장된 모델을 제공한다. 먼저 length와 use를 사용하여 최초 발생부터 몇 번째 이벤트 인스턴스까지 사용하고 이후는 폐기할 것인 지 결정할 수 있다. 예를 들어 length(50), use(first)로 지정하면 첫 번째 발생한 인스턴스부터 50번째까지 사용하고 이후에 발생하는 인스턴스들은 폐기함을 의미한다. 특정 이벤트 인스턴스를 지칭 시에는 현재 작성하고 있는 이벤트 인스턴스는 this로 가리킬 수 있고 prev(k)를 이용하여 k번째 이전의 인스턴스를 가리킬 수 있다. 파라미터 없이 (즉 prev로) 사용한 경우는 prev(1)와 같다. 이처럼 이벤트 이름에 인덱스를 사용하여 예전의 인스턴스를 사용할 수 있다. 만일 인덱스를 생략하면 가장 최근에 검출된 인

스턴스를 지칭한다. 이벤트 인스턴스 앞에는 @ 연산자를 사용하여 그 이벤트 인스턴스의 타임스탬프를 돌려받을 수 있다. 예를 들어 @prev(1)은 동종의 복합이벤트 중 직전에 일어난 인스턴스의 타임스탬프를 나타낸다. R(Relative) 연산자는 지정된 시간을 기준으로 그때까지 발생된 이벤트들 중 최근 인스턴스를 골라낼 수 있게 한다. 여기에 @ 연산자를 붙여서 그 인스턴스의 타임스탬프를 얻어낼 수도 있다.

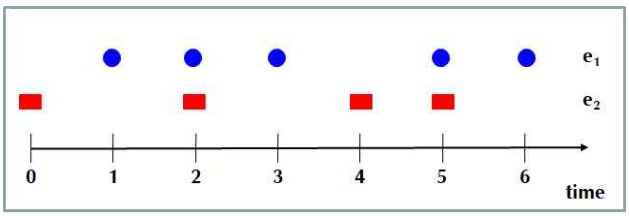

Fig. 4. Event Instance creation

그림 4는 어떤 시스템에서 두 종류의 이벤트가 발생한 연혁(event occurrence history)를 보여준다. 예를 들어 이벤트 e1은 시스템 시간 1, 2, 3, 5, 6에 검출되었고, 각각의 발생에 대해 이벤트 인스턴스가 생성된다. 앞에 제시된 표기법을 이용하면 @(e1, 1) = 1, @(e1, 2) = 2, @(e1, 3) = 3, @(e1, 4) = 5, @(e1, 5) = 6이다. R(e2, @e1, 3)은 e1의 가장 최근 인스턴스의 타임스탬프를 기준으로 3번째 이전의 e2 타입의 인스턴스를 가리킨다. 따라서 @R(e2, @ e1, 3) = 2이다.

• 시간 제약 연산자

기존 연구에서는 이벤트가 발생한 특정 지정 시간에 대한 연산만 고려하였다. 본 논문에서는 지정 시간외에 간격 시간을 지정할 수 있도록 연산자를 확장한다. 그림 5에서 이벤트 e1의 발생이 완료된 시간 t2가 e1의 지정 시간이고, [t1, t2]는 e1의 간격 시간이다. 즉, 정의하면 다음과 같다. T(e)가 이벤트 e의 시간을 나타낼 때, Tb(e)는 시작 시간(begin time)을, Te(e)는 완료 시간(ending time)을 가리킨다. 일반적으로, 기본 이벤트는 지정 시간을 사용하고 복합 이벤트는 간격 시간을 사용한다.

본 논문에서는 서로 다른 이벤트들의 간격 시간들이 서로 독립적일 때 강한 시간 관계를 가졌다고 하며, 서로 다른 이벤트들의 간격 시간들이 겹쳐질(overlapped) 수 있을 때 약한 시간 관계를 가지고 있다고 하고, 다음과 같이 정형적으로 정의한다.

정의 4: 서로 다른 이벤트 사이의 강한 시간 관계는 다음과 같이 정의할 수 있다.

$$T(e1) < T(e2) \Leftrightarrow Te(e1) < Tb(e2)$$

정의 5: 서로 다른 이벤트 사이의 약한 시간 관계는 다음과 같이 정의할 수 있다.

$$T(e1) \prec T(e2) \Leftrightarrow$$
$$Tb(e1) < Tb(e2) \wedge$$
$$\{(Te(e1) > Te(e2)) \vee (Te(e1) < Te(e2))\}$$

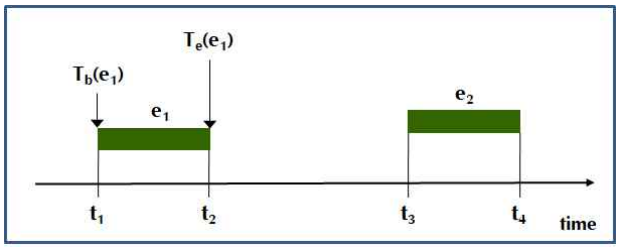

Fig. 5. Point time and Interval time

• 인과관계 연산자

이벤트 인과관계란 발생된 이벤트들 사이의 의존성을 의미한다. 즉, 먼저 발생 한 이벤트가 이후 발생한 이벤트의 원인이 되는 관계로, 본 논문에서는 직접 인과관계와 간접 인과관계로 분류된다. 직접 인과관계란 원인이 된 이벤트와 결과 이벤트 사이에 중간에 발생한 이벤트가 없는 경우를 의미하고, 간접 인과관계란 원인 이벤트와 결과 이벤트 사이에 중간 이벤트들이 존재하는 경우를 의미한다. 이를 정형적으로 정의하면 다음과 같다.

정의 6: 직접 인과관계는 → 연산자를 사용하여 다음과 같이 정의할 수 있다.

$$e1 \rightarrow e2 \Rightarrow e1 \in e2.c$$

정의 7: 간접 인과관계는 $\dashrightarrow$ 연산자를 사용하여 다음과 같이 정의할 수 있다.

$$e1 \dashrightarrow e2 \Rightarrow$$
$$\exists ei, I \neq 1, 2, e1 \rightarrow ei \wedge ei \rightarrow e2$$

기존 연구[8, 9]에서는 이벤트들의 시간적 선후관계만 다루어 인과관계를 별도의 요소로 다루지 않았다.

5. Applying Event Operators

복합 서비스 (Composite Service)인 기업의 비즈니스 프로세스는 하나 이상의 서브 프로세스들로 구성되며 서브 프로세스 역시 더 낮은 단계의 서브 프로세스들로 구성될 수 있다. 이러한 경우 하나의 서브 프로세스 상의 오류가 관계있는 다른 서브 프로세스나 다른 단계의 서브 프로세스의 실행에도 영향을 주어 다량의 관련 이벤트가 발생되기도 한다[3]. 이러한 경우에 이벤트 계층 구조와 각종 이벤트 연산자가 관리자로 하여금 "이벤트 홍수"에 빠지지 않고 정확히 진단에 필요한 이벤트

만을 필터링 할 수 있는 기능을 제공한다. 예를 들어 다음의 복합 이벤트 CascadingFailure는 가장 하위 계층의 서브 프로세스 오류가 파급되어 2단계 및 1단계 서브 프로세스의 오작동이 감지된 경우에 발생된다.

```
Complex CascadingFailure {
 Pattern ( SEQ( SPFailureLevel3,
 SPFailureLevel2, SPFailureLevel1, 5min))
}
```

다음은 어떤 링크에 이상이 생기면 자동적으로 그 링크에 대한 복구 노력이 행해지는 시스템의 경우, 자동 복구 시스템이 잘못 되었을 상황을 감지해서 이벤트를 발생하는 예이다. 단 링크 이상에 관한 이벤트 중 SystemAllShutDown이벤트는 제외한다.

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

추가적으로 다음과 같은 이벤트를 정의해서 위의 AutoRepairNotOccurWarning가 일어나면 최대 10분에 한 번씩 관리자에게 리포트하는 행위를 추가할 수 있다.

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

본 연구의 응용은 센서 데이터의 이벤트 처리에 제한되지 않고 이벤트 기반의 컴퓨터 시스템들의 다양한 경우에 적용될 수 있다. 예를 들어 다음의 예제는 인터넷 기반의 전자상거래 시스템에서 소비자가 물건을 찾아 장바구니에 옮기고 대금지불을 하는 구매 절차를 1분 안에 마치는 이벤트를 검출한다. 만일 이러한 패턴이 동일한 물건에 대해 반복된다면 FrequentSamePurchase 이벤트를 발생시킨다. 이러한 복합 이벤트가 짧은 시간 안에 자주 일어난다면 이것은 시스템의 오류로 인해 구매가 비정상적으로 폭증한 것으로 의심해 볼 수 있다.

```
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))
 }
```

6. Event context

이벤트 컨텍스트는 보다 정교한 다중 상황 검출을 위해 낮은 단계의 이벤트에서 높은 단계의 정보로 변환되는 과정에서 필요한 부가적인 정보를 정의할 수 있도록 사용된다. 또한, 이벤트 인스턴스를 분류하는 역할을 하며 의미 관점과 추상화 계층의 두 가지 요소로 구성된다.

의미 관점은 AOP의 횡단 관심사에 해당하는 관점을 추가함으로써 다중 상황 검출을 가능케 하는 일시적인 컨텍스트이다. 일시적이라는 말은 의미 관점이 여러 상황들을 포함하는 시간 윈도우이기 때문이다. 의미 관점은 이벤트 데이터와는 비교적 독립적인 컨텍스트 정보, 예를 들면 그림 6에서 볼 수 있는 것처럼 location(위치), role(역할), state(상태) 등의 요소를 포함하며 initiator(개시 이벤트)와 terminator(종료 이벤트)로 명명되는 두 가지 이벤트를 발생시켜 시작과 종료의 경계를 설정한다. 즉, initiator의 발생은 의미 관점을 초기화하는 역할을 하고, terminator의 발생은 의미 관점을 종료시키는 역할을 한다. 그림 6의 의미 관점 정의에서 initiator와 terminator의 condition에 의해 정의된 조건이 충족될 경우 의미 관점이 각각 생성 및 종료된다.

추상화 계층은 의미 관점에서 정의된 속성에 대하여 그 특성을 반영한 범위를 계층 구조로 정의한다. 정의된 속성은 복합 이벤트의 성격에 따라 위치, 조직, 제품 등과 같이 서로 다른 규격을 적용하는 대상들일 수 있다. 예를 들어 그림 7에서 의미 관점의 location에 정의된 "Shop Floor(생산 현장)"라는 하나의 항목은 위치에 대하여 높은 단계에서 낮은 단계로 계층화하여 표현할 수 있다. 즉, 이벤트를 수집하는 "reader"가 최하위 단계이고 "shop floor"가 가장 높은 단계로 정의되어 있다. 이와 같이 추상화 계층을 사용함으로써 다양한 이벤트들을 처리하는 과정에서 이벤트 속성들의 추상화가 가능하고 이를 통해 이벤트들을 분류하기가 용이해진다. 또한, 앞서 기술한 인과관계 벡터, 인과관계 연산자와 함께 사용하여 원인-결과 관계를 도출하여 이벤트간의 행위 분석을 용이하게 할 수 있다. 즉, 관심사항에 대하여 근본 원인이 된 이벤트와 해당 이벤트의 최하위 계층의 이벤트 소스까지 검출해 냄으로써 정교한 이벤트 분석을 가능하게 한다.

```
<SemanticAspect name="work shift">
  <initiator event="shift transition">
    <condition name="executeCondition"
      type="tSourceCondition">
  </initiator>
  <location name="shop floor"/>
  <role name="worker"/>
  <state name="production"/>
  <terminator event="shift transition">
    <condition name="completion"/>
  </terminator>
</SemanticAspect>
```

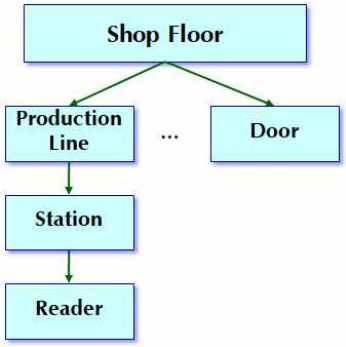

Fig. 6. Semantic Aspect and Abstract hierarchy

V. Case Study

본 장에서는 제안된 복합 이벤트 모델의 적용성 및 응용성을 보여주기 위해서 제조업체의 냉장고 생산 라인의 비즈니스 프로세스 모델을 적용한다. 먼저 적용할 시나리오에 대하여 기술하고, 시나리오에서 발생할 수 있는 주목할 만한 상황을 제안된 이벤트 모델을 이용하여 명세한 후, 응용 사례 연구의 결과로 장점을 기술한다.

Table 1. Refrigerator produce process scenario

(1)The refrigerator body and pre-packed liners are concurrently prepared and when both are ready, the process begins. (2)The worker assembles the body such as the base plate and back plate of the refrigerator, (3)The foam is formed and dispersed in a polymer resin to form a product, (4)Incorporate the appropriate compressor for each model to assemble the refrigerator and compressor. (5)Check items such as cell size, shape, and distribution from the bubbles generated in the foam molding process. If the quality is not appropriate, the refrigerator returns to operation (3). (6)Finally, wrap the package using the prepared packing lining.

1. Scenario

적용될 시나리오는 한 가전업체의 냉장고 생산 공장의 제조 업무로 RFID 리더기가 각 생산 라인의 주요 위치에 설치되어 있고 이를 통하여 이벤트를 수집한다고 가정한다. 시나리오의 구성은 표 1과 같다.

그림 7은 표 1의 시나리오를 적용하여 복합 서비스인 비즈니스 프로세스 형태로 표현한 것이다. 비즈니스 프로세스 모델링은 BPEL 2.0 표준을 준수하는 오픈 소스인 uEngine BPM [15]을 사용하였다.

2. Specification of complex event

앞서 정의한 시나리오에 대하여 운용과 관리 두 가지 측면에서 본 연구에서 제안된 복합 이벤트 모델을 응용해본다.

첫째, 먼저 조립 공정에서 부품과 냉장고간의 잘못된 조립 등과 같은 운용측면에서 응용할 수 있다. 예를 들어 다음은 “압축기 조립 활동(Condensation Assembly Activity)”에서 냉장고와 압축기 1대를 조립 시 RFID 리더기 등 센서에 의해 냉장고 및 압축기의 종류가 식별되고 만약 잘못된 압축기가 결합되고 있고 이벤트 간격 시간이 10분일 경우 작업요원에게 잘못된 압축기 종류가 조립되고 있다는 것을 알려주기 위한 복합 이벤트를 검출한다. 조건절의 CONTEXT 키워드 뒤에 명시한 “작업 교대(work shift)”라는 이벤트 컨텍스트 정보는 의미 관점에 포함된 위치, 역할, 상태 정보와 추상화 계층의 계층 구조로 정의된 위치 정보를 통해서 해당 이벤트가 생산 현장에서 작업자가 생산 공정 작업 중 발생한 것이며, 어떤 작업대의 어떤 리더기를 통해 발생하였는지도 알 수 있다. 즉, 근본 원인이 있는 장소를 정확하게 지적할 수 있다. 본 시나리오에 대한 이벤트 컨텍스트의 구체적인 항목별 데이터 값은 3.6절의 그림 6에 정의되어있다.

```
Complex TypeCheck {
 Pattern (A=AND (REFRIGERATOR,
   OR (COMPRESSOR01, COMPRESSOR02)))
 Where ([compressor_type] and A == NULL
        CONTEXT work shift INTERVAL
        10 min)
}
```

둘째, 작업 시간 및 품질 관리, 배송 서비스 관리 등과 같은 관리측면에서 응용할 수 있다. 다음은 작업시간과 품질을 정확하게 평가하기 위해 기본 이벤트인 WORKTIME과 QUALITY의 기록에 따라 모든 작업요원의 모든 작업장에서 작업시간과 품질을 분석할 수 있는 복합 이벤트이다.

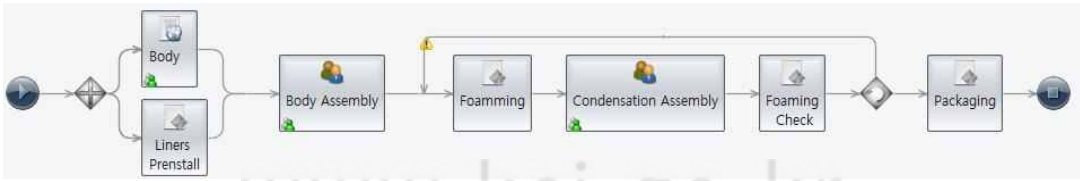

Fig. 7. A business process model for Refrigerator produce process

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

다음은 제품의 배송 서비스 중 배송 착오를 검출할 수 있는 예로, TRUCK 이벤트는 배송 수단인 트럭이 준비되었음을 명세하고, EXIT-READING 이벤트는 context.type의 내용을 통해 올바른 제품과 수요자인지를 확인하고 잘못 되었을 경우 검출되는 복합 이벤트이다.

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

본 연구의 응용은 기업의 서비스 관리에만 제한되지 않고 다양한 경우에 적용될 수 있다. 예를 들어 다음은 최근 컴퓨터 네트워크 기반의 시스템에 자주 발생하고 있는 DOS(Denial of Service) 공격의 검출 예이다. TCP/IP 프로토콜의 Sync 및 Ack 메시지에 대응되는 SynEvent와 AckEvent를 이용하여 SyncEvent가 AckEvent 없이 같은 소스에서 연속되는 경우 FrequentHalfOpen이벤트를 발생시키고, 이러한 Frequent-HalfOpen 이벤트가 짧은 시간 안에 반복되는 경우 DOS 공격에 대한 경보를 발생시킨다.

```
Complex FrequentHalfOpen {
 Pattern (SEQ(SynEvent as x, ~AckEvent, SynEvent
          as y, 5sec))
 Where (@this - @prev <= 10sec and x["source"] =
          y["source"])
}
Complex DosSymptom {
 Pattern (COUNT(FrequentHalfOpen , ">=", 10, 2min))
 Action Call( Message.SendDOSWarning())
}
```

앞서 정의한 시나리오에 대하여 제안된 이벤트 모델을 응용해 본 결과 수집된 이벤트들의 조합의 의미를 발견하고 이를 해석하여 기업에 보다 가치 있고 유용한 비즈니스 정보의 제공이 가능하다는 것을 알 수 있었다. 이를 운용과 관리 측면으로 나누어 구체적으로 살펴보면 다음과 같다.

1. 운용 측면에서 이벤트들간의 시간, 인과관계 및 계층 관계를 추출해 낼 수 있다. 이 기능은 이벤트간 혹은 시스템의 행위 분석을 용이하게 할 수 있다. 또한, 이벤트 컨텍스트 매커니즘은 이들 연산자와 함께 사용되어 관심사항에 대하여 근본 원인이 된 이벤트와 해당 이벤트의 최하위 계층의 이벤트 소스까지 검출해 냄으로써 정교한 이벤트 분석을 가능하게 한다. 이에 대한 예는 TypeCheck 복합 이벤트에서 구체적으로 살펴볼 수 있었다.

2. 관리 측면에서 관리자들의 정확한 판단과 조기 대응을 가능하게 할 수 있다. 위 사례 중 배송 서비스의 예를 들면, 과거의 제품 배송 이력을 분석하여 배송 지연의 문제를 분석하고 현재 진행중인 배송 현황을 개선하고자 할 때, 기존의 방법은 배송 시간에 영향을 미치는 지표를 정성적으로 탐색하여 비즈니스 규칙을 적용하는 정성적 접근법을 사용하였다. 이러한 방법의 경우 예기치 못한 상황이나 새로운 추세를 민첩하게 반영하지 못할 우려가 있다. 본 연구에서 제안된 이벤트 모델은 의사 결정에 필요한 정보를 지원하며, 이상 징후의 검출을 자동화하여, 변화에 능동적으로 대응하면서 이상 징후를 조기 경보 할 수 있다는 장점이 있다.

3. 관리 측면에서 궁극적으로 자동화된 엔터프라이즈 서비스 모니터링을 가능케 한다. 이상 징후 혹은 다중 상황 검출의 자동화와 더 나아가 이에 대한 대응 역시 복합 이벤트의 처리절에 명시함으로써 자동화된 대응 및 조치를 가능케 한다. 이에 대한 예는 FrequentHalfOpen 복합 이벤트에서 살펴볼 수 있었다. 대응은 단순한 경보 발생에서부터 엔터프라이즈 시스템의 특정 기능을 작동시키거나 BPMS의 특정 비즈니스 프로세스를 실행 시키는 등으로 확장될 수 있다.

VI. Assessment

기존 연구와 비교해 보면, McGregor[8]는 이벤트 프로세싱 컨테이너를 통하여 실시간에 많은 수의 프로세스 이벤트들을 처리할 수 있도록 하였다. 하지만 기반 이벤트 모델이 단순 이벤트 모델로 복합 이벤트를 지원하지 않아 단순히 알람(alarm) 이벤트의 정적인 포함 관계 기술에 그치고 있다. Wang[10]은 실시간 시스템의 시간적 양태(temporal behavior)를 기술하기 위해 개발된 Real-Time Logic(RTL)[11]을 확장한 복합 이벤트 모델에 기반하고 있다. 따라서 이벤트 인스턴스 수준의 명세가 가능하고, 단순 계층 지원 연산 및 인과관계 명세를 지원한다. 하지만, 네트워크 관리 시스템이라는 특정 도메인에 초점을 맞추어 제한적인 이벤트 명세만 제공함으로써 복잡한 비즈니스 환경의 적용과 표현에는 한계가 있다. Luckham[9]의 연구에서는 복합 이벤트 구현을 위해 RAPIDE 언어를 개발하였으며 이벤트의 연관성 정보를 사용하여 이벤트 패턴, 추출, 결합 등의 기술을 사용한 이벤트 모델을 제안하였다. 하지만 수집되는 이벤트의 메타모델이나 구체적인 형태는 언급하지 않았으며,

이벤트 타입에 기반한 모델이기 때문에 인스턴스 간의 복잡한 시간 관계를 나타내는 데에는 부족함이 있다.

본 연구에서 제안된 이벤트 모델을 평가하기 위하여 다음과 같은 평가항목을 선정하였다. 선정 기준은 이벤트 모델의 표현력을 결정짓는 연산자의 다양성을 기반으로 관련 연구[5, 8, 9, 10]에서 중요하게 언급된 이벤트 계층 및 시간적 관계, 인과관계에 관련된 연산자 항목과 정교한 이벤트 분석을 가능하게 하는 기법 등을 평가 항목으로 선정하였다. 이를 통하여 제안된 이벤트 모델의 특장점을 알 수 있다.

- 기반 이벤트 모델: 이벤트 처리 과정의 기반이 되는 이벤트 모델이 낮은 단계의 단순 이벤트 처리에 기반한 단순 이벤트 모델인지 높은 단계의 정보로 변환되는 복합 이벤트 처리 기반의 복합 이벤트 모델인지에 대한 평가

- 이벤트의 정형 명세: 이벤트 인스턴스들의 클래스에 해당하는 이벤트 타입을 정형 명세 기법을 사용하여 정의함으로써 모호성 없는 이벤트 명세를 지원하며, 이를 통한 이벤트 처리의 확실한 기초를 제공하는지에 대한 평가

- 이벤트 타입에 의한 명세: 이벤트를 정의 또는 명세할 때 이벤트 타입에 의한 명세를 지원하는 지에 대한 평가

- 이벤트 인스턴스 수준의 명세: 이벤트 타입은 많은 이벤트 인스턴스를 가지고 있을 수 있다. 이벤트 타입으로만 명세에 표시하면, 같은 타입의 복수개의 인스턴스가 존재할 때, 복합 이벤트의 검출 시 후보가 되는 이벤트의 인스턴스 중 어느 인스턴스를 사용하며 어떻게 이력 버퍼에서 삭제할 지에 대한 해결이 필요하다. 따라서 특정 인스턴스를 지정하는 인스턴스 지칭 연산자가 필요하다.

- 이벤트 계층 표현: 이벤트를 계층적으로 구조화 시키는 것은 복잡성을 컨트롤하기 위한 디자인 기술로 동등 계층 및 상/하 계층의 이벤트 간에 서로 연관성을 정의해 줌으로써 복잡성을 단순화 시켜줄 수 있으며, 복잡한 비즈니스 환경을 적용 및 표현하기 위해 필요하다.

- 확장된 시간 관계 표현: 실시간 모니터링을 위해서는 발생되는 이벤트 인스턴스 간의 정확한 타이밍 제약에 관한 명세가 가능해야 하며 이를 위해 지정 시간에 대한 연산뿐만 아닌 시간적 관계를 원활하게 표현할 수 있는 연산자가 필요하다.

- 인과관계 명세: 이벤트간의 원인-결과 관계를 도출하여 이벤트간 혹은 시스템의 행위 분석을 용이하게 할 수 있는 인과관계 명세를 지원하는지에 대한 평가

- 정교한 다중 상황 검출 기법: 다양한 연산자와 규칙, 기능을 제공하는 것 외에 이벤트 데이터와는 독립적으로 더 많은 상황 정보를 검출하며 보다 정교한 이벤트 분석을 가능하게 하는 기법을 제공하는지에 대한 평가(본 논문에서는 이벤트 컨텍스트 등)

이상의 평가 항목으로 표 2에서와 같이 기존 연구와 본 논문을 비교해 보았다.

Table 2. Comparison with related research

Technology / Criterion	McGregor	Luckham	Wang	Proposed Technique
Support Complex Event Model		O	RTL	O
Providing Formal Specification for Event			△	O
Support Specification for Event type	△	O	△	O
Support Specification for Event instance level			O	O
Support Specification for Event hierarchy structure		△	△	O
Support Specification for extended time correlation (Interval time)				O
Support Specification for causality			△	O
Providing multiple situation detection method				O

VII. Conclusions

본 연구에서는 기존에 연구되었던 다양한 이벤트 모델들을 확장하고, AOP의 장점을 통합하여 다양한 이벤트 소스에서 발생하는 이벤트들을 효과적으로 처리하여 다중 상황 검출을 가능케 하는 확장된 복합 이벤트 모델을 제시하였다. 제안된 이벤트 모델은 다양하고 풍부한 연산자와 규칙, 기능 및 보다 정교한 이벤트 분석을 위한 기법 등을 제공한다. 본 연구의 결과를 요약하면 다음과 같다.

첫째, 먼저 이벤트 타입을 정형 명세 기법을 사용하여 정의함으로써 모호성 없는 이벤트 명세를 지원하며, 이를 통한 이벤트 처리의 확실한 기초를 제공하였다.

둘째, 이벤트 타입뿐만 아니라 이벤트 인스턴스 단위로도 이벤트 명세를 할 수 있도록 인스턴스 지칭 연산자를 제공함으로써 복잡한

비즈니스 환경의 적용을 위한 모델의 표현력을 증가시켰다.

셋째, 이벤트의 계층적 구조화를 지원하여 동등 계층 및 상/하 계층의 이벤트 간에 서로 연관성을 정의해 줌으로써 복잡성을 단순화 시키며, 복잡한 비즈니스 환경의 적용 및 표현을 용이하게 하였다.

넷째, 발생되는 이벤트 인스턴스 간의 정확한 타이밍 제약에 관한 명세를 위해 지정 시간외에 간격 시간을 지정할 수 있도록 연산자를 확장하였으며, 이벤트 간의 원인-결과 관계를 도출하여 이벤트간 의 행위 분석을 용이하게 하는 인과관계 연산자를 제공하였다.

다섯째, 이벤트 데이터와는 독립적으로 더 많은 상황 정보를 검출하며 보다 정교한 이벤트 분석을 가능하게 하는 이벤트 컨텍스트 매커니즘을 제공하였다.

이와 같이 설계된 이벤트 모델로써 다양한 사례에 연산자를 적용하여 보았고, 구체적인 응용 사례를 통하여 적용 가능성 및 유효성을 확인할 수 있었다. 마지막으로 기존의 연구들에 비해 본 연구에서 제안된 모델이 가지는 장점을 제시하였다.

본 연구는 향후에 추가적인 연구가 필요하다고 판단된다. 이를 구체적으로 살펴보면 첫째, 제안된 이벤트 모델은 이벤트들 간의 계층적 구조를 지원하여, 일반적인 복합 이벤트 시스템에 비해 수행 시간 부담이 예상된다. 또한 이벤트 인스턴스에 대한 연산자 지원 및 시간 관계 연산자 처리에 대해 최적화 전략이 고려되어야 할 것이다. 둘째, 최적화 전략과 제안된 이벤트 모델의 이벤트 검출 알고리즘 구현 및 실제 시스템에 적용해 보는 것이 필요할 것이다. 셋째, 구현한 프로토타입을 통해 정량적인 성능 측정도 필요할 것이다. 성능 측정 결과를 통해 증진된 표현력과 수행 시간 부담에 대한 분석을 할 수 있을 것이다.

REFERENCES

[1] N. Y. Lee and C. R. Litecky, "An empirical study of software reuse with Special Attention to Ada," IEEE Trans. on Software Engineering, Vol. 23, No. 9, pp. 537-549, September 1997.

[2] K. M. Chandy, "Event-Driven Applications: Costs, Benefits and Design Approaches," California Institute of Technology, November 2006.

[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] S. Chakravarthy et al., "Composite Events forActive Databases: Semantics, Contexts and Detection," In Proc. of the International Conference on Very Large Data Bases (VLDB), Vol. 9, No. 6, pp. 606-617, September 1994.

[5] G. Liu et al., "Composite Events for Network Event Correlation," Proc. of the IFIP/IEEE Symposium on Integrated Network Management, May 1999.

[6] U. Dayal et al., "The HiPAC Project: Combining Active Databases and Timing Constraints," ACM SIGMOD RECORD, Vol. 17, No. 1, pp. 51-70, March 1988.

[7] G. Kiczales et al., "Aspect-oriented programming," In Proc. of the European Conference on Object-Oriented Programming (ECOOP), Vol. 1241, No 1, pp. 220-242, June 1997.

[8] 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.

[9] D. Luckham and B. Frasca, "Complex Event Processing in Distributed System," Stanford University Tech, Report CSL-TR-98-754, Mar, 1998.

[10] 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.

[11] 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.

[12] N. W. Paton and O. Diaz, "Active Database Systems," ACM Computing Surveys, 31(1), 1999.

[13] A. K. Mok et al., "Specifying Timing Constraints and Composite Events: An Application in the Design of Electronic Brokerages," IEEE Transactions on Software Engineering, Vol. 30, No. 12, pp. 841-858, Dec 2004.

[14] C. G. Lee et al., "Monitoring of Timing Constraints with Confidence Threshold Requirements," IEEE Transactions on Computers, Vol. 56, No. 7, pp. 606-617, May 2007.

[15] uEngine BPM, http://uengine.org/

Authors

Deuk Kyu Kum received the M.S. and Ph.D. degrees in Computer Science and Engineering from Soongsil University, Korea, in 2005, 2012, respectively Dr. Kum joined the faculty of the Department of Information Technology at Yuhan University, Bucheon-si, Gyeonggi-do, Korea, in 2017. He is currently a Professor in the Department of Information Technology, Yuhan University. He is interested in big data analysis technology, internet and mobile computing, and cloud computing.