이벤트 상관 기반의 네트워크 관리 시스템을 위한 복합 이벤트 모델의 설계

(The Design of an Extended Complex Event Model for the Event Correlation Based Network Management Systems)

이 기 성 † 이 창 하 †† 이 찬 근 †††
(Ki-seong Lee) (Chang Ha Lee) (Chan-gun Lee)

요 약 본 연구에서 우리는 복합 이벤트(complex event)와 관점지향 프로그래밍(aspect-oriented programming)을 함께 고려하여 확장된 복합 이벤트 모델을 제시한다. 우리는 이 두 모델의 통합을 통해 이벤트 상관 기반의 네트워크 관리 시스템에 적합한 진보된 이벤트 명세 방법을 제안한다. 구체적으로, 계층적 이벤트 구조를 지원하도록 모델을 확장하고 관점지향 프로그래밍의 교차점(point cut)을 이벤트로 인식하도록 한다. 또한 이벤트 명세를 인스턴스(instance) 단위로 할 수 있도록 이벤트 연산자를 제공하고 시간적 관계를 원활하게 표현할 수 있도록 한다. 마지막으로 다른 이벤트 모델과의 비교를 통해 본 이벤트 모델의 장점을 제시한다.

키워드 : 관점지향 프로그래밍, 복합 이벤트, 계층적 구조, 이벤트 명세, 시간적 관계, 네트워크 관리

Abstract In this study, we present an extended complex event model by considering both of the complex event and the aspect-oriented programming. We propose an advanced scheme for the event specification suited for the event correlation based network management systems by merging these two models. 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. Lastly, we compare the proposed model with other event models and present the benefits of it.

Key words : aspect-oriented programming, complex event, hierarchical event structures, event specification, temporal relation, network management

1. 서 론

네트워크 관리 시스템은 컴퓨터 네트워크를 구성하고 있는 장비들에 대해 중앙 감시 기능을 제공한다. 최근 컴퓨터 네트워크에서 발생할 수 있는 다양한 문제들에 대해 체계적으로 분석하고 능동적으로 대처할 수 있는 방법에 대한 연구가 활발히 진행되었다. 네크워크 관리 시스템에서 이벤트는 시스템의 상태 변화(change of status)를 의미한다. 예를 들어 네크워크의 고장과 직접적으로 관련될 수 있는 특정 노드의 무 반응, 링크의 소실, 장비에서의 알람(alarm) 등은 모두 이벤트로 표현되어 네크워크 관리 시스템의 입력이 된다. 이외에도 갑작스런 성능의 저하, 보안에 관한 위반 등 간접적으로 네크워크의 고장과 관련이 있는 이벤트도 존재한다.

이벤트 상관에 기반한 네크워크 관리시스템은 이와 같은 다양한 이벤트들에 대한 개별적인 감시 기능뿐 아니라 그들간의 상관 관계에 대한 종합적인 질의 및 필터 기능을 제공하여 실제로 어떠한 원인 때문에 그러한 네트워크 이상이 발생했는지 분석하는데 효과적이다.

복합 이벤트(complex event)[1]는 이벤트들간의 연

· 본 연구는 서울시 산학연 협력사업(과제번호 SJ080658)과 한국연구재단 기초연구사업(과제번호 331-2008-1-D00452, 2009-0070392)의 지원을 받았습니다.

† 학생회원 : 중앙대학교 컴퓨터공학부
goory00@gmail.com
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
chlee@cau.ac.kr
††† 정 회 원 : 중앙대학교 컴퓨터공학부 교수
cglee@cau.ac.kr
(Corresponding author임)

논문접수 : 2009년 9월 7일
심사완료 : 2009년 10월 15일

정보과학회논문지: 정보통신 제37권 제1호(2010.2)

관 관계를 이용해서 정의되며, 특정 상황에서 자동적으로 적절한 반응을 보여야 하는 시스템들에서 이벤트-조건-행동(event-condition-action, 이하 ECA) 규칙을 이용한 체계적인 구현 방법을 제공한다.

우리는 본 연구에서 복합 이벤트와 관점지향 프로그래밍(aspect-oriented programming, 이하 AOP)[2]을 함께 고려하여 확장된 복합 이벤트 모델을 제시하고, 제안된 이벤트 모델이 어떻게 이벤트 상관에 기반한 네트워크 관리 시스템에 응용될 수 있을지 보인다. AOP는 기존의 프로그래밍 방식보다 진보된 모듈화 기능을 제공하여 시스템의 유지 보수에 도움을 준다. AOP는 결합점(join point) 및 교차점(point cut)의 개념을 제공하며 이들은 이벤트에 기반한 프로그래밍과 밀접한 관련이 있다.

구체적으로 우리는 본 연구에서 AOP의 교차점을 일반적인 복합 이벤트로 인식하도록 하고, 클래스나 관점(aspect)에 독립적인 복합 이벤트 계층을 정의할 수 있도록 복합 이벤트의 정의 방법을 확장한다. 또한 이벤트 타입 뿐만 아니라 이벤트 인스턴스 단위로도 이벤트 명세를 할 수 있도록 인스턴스 지칭 연산자를 제공한다. 그리고 실시간 시스템 모니터링 분야에서 연구된 결과를 적용하여 이벤트 인스턴스간에 시간적 관계를 원활하게 표현할 수 있게 연산자를 확장한다. 마지막으로 네트웍 관리 시스템 에서 등장할 수 있는 시나리오들을 제안된 이벤트 모델로 기술하여 본 연구의 효과를 보인다.

본 논문은 다음과 같이 구성되어 있다. 2장은 AOP 및 복합 이벤트에 대한 기존의 연구 성과를 요약한다. 3장에서는 기존의 AOP와 복합 이벤트 모델에 대한 통합의 필요성과 방법을 기술한다. 또한 기존 모델에 비해 확장된 이벤트 연산을 소개하고 구체적인 연산자들을 정의한다. 4장에서는 앞서 제시된 이벤트 모델을 이용하여 복합 이벤트의 정의 방법을 제시하고 실제로 복합 이벤트들을 정의한 예를 살핀다. 또한 기존 시스템들과의 표현력 비교를 통해 제안된 이벤트 모델의 우수성을 보인다. 5장은 논문을 요약하며 향후 연구를 제시한다.

2. 관련연구

이벤트 기반 시스템에서는 시스템의 상태 변화를 이벤트로 표현한다. 이벤트는 원시 이벤트, 기본 이벤트, 복합 이벤트로 분류할 수 있다. 원시 이벤트(raw event)는 논리적 혹은 물리적 시스템에서 검출 되었으나 아직은 시스템에서 정식으로 사용 되기 전의 비가공 이벤트를 말한다. 원시 이벤트에는 노이즈가 포함되어 있거나 타임스탬프 등 처리에 필요한 필수 요소들이 결여되어 있을 수 있기 때문에 원시 이벤트 필터 등을 이용하여 기본(primitive) 이벤트로 변환된다.

기본 이벤트는 시스템에서 사용될 수 있는 이벤트이며 더 이상 잘게 분리할 수 없는 단위 요소이다. 기본 이벤트에는 노이즈와 중복성이 없으나 시스템의 매우 단순한 상태 변화를 의미하기 때문에 조금 더 구체적이고 의미가 있는 복합 이벤트가 필요하다. 복합 이벤트는 기본 이벤트들과 다른 복합이벤트를 이용하여 정의되는데 이때 이벤트 모델에서 제공되는 이벤트 연산자(event operator)에 따라 그 표현력(expressiveness)이 달라진다.

능동 데이터베이스(active database)[3] 혹은 복합 이벤트 관리 시스템(complex event management system) 등은 ECA 규칙을 사용하여 시스템에서 벌어지는 상황에 따라 자동적으로 반응하도록 설계된다. 이벤트 절은 위에 기술된 복합 이벤트의 형식으로 표현되며 조건절에는 부가적인 조건들을 기술한다. 시스템의 운영 중에 이벤트가 발생하고 조건이 만족되면 해당하는 규칙에 명시된 행동이 취해지게 된다. ECA 규칙을 채용한 대표적인 시스템에는 Snoop[4], HiPAC[5], Ode[6] 등이 있다.

이벤트 상관에 기반한 네트워크 관리 시스템의 개발은 90년대 초반부터 활발히 이루어졌다. 대표적인 초기 시스템에는 IMPACT[7], NetFACT[8], InCharge[9] 등이 있다. 이들은 하나의 단순한 네트워크 고장에 의해서도 과도하게 많은 관련 이벤트가 발생되어 오히려 네트워크 관리자들이 올바른 원인을 찾기가 어렵다는 점에 착안하였다. 이벤트 계층에 대한 지원과 함께 이벤트간의 원인-결과 관계를 나타낼 수 있도록 하여 정제된 상황 파악을 통해 원인 분석을 보다 쉽게 할 수 있도록 하였다. 또한 OR, COUNT 등과 같이 비교적 단순한 이벤트 연산자들을 지원하여 효과적인 이벤트 필터링을 수행할 수 있도록 하였다.

본격적인 복합 이벤트를 지원하는 네트워크 관리 시스템으로는 Hasan[10], Liu[11] 등의 연구가 있다. Hasan[10]은 이벤트의 연관을 효과적으로 지원하기 위해 능동 데이터베이스의 복합이벤트의 사용을 제안하였다. 이후에 Liu[11] 등은 이벤트의 타입에만 의존하는 복합이벤트 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)[12]의 이벤트 모델을 확장하여 이벤트 인스턴스에 기반한 복합 이벤트 정의 방법과 관련 이벤트 연산자를 정의하였다.

최근 등장한 AOP는 기존의 구조적 프로그래밍(structured programming) 및 객체지향 프로그래밍(object-oriented programming, 이하 OOP) 언어들이 공통적으

로 제일 관심사(primary concern) 이외의 횡단관심사(cross-cutting concerns)들에 대해서는 모듈화 할 수 있는 방법을 제공하지 못한다는 문제점의 인식에서 출발한다. AOP는 OOP의 대체가 아니라 확장의 개념이며, 관점이라는 도구를 통해 횡단 관심사들을 모듈화한다. 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.

최근 Cilia[13] 등은 능동 데이터베이스와 AOP가 공통적으로 해당 시스템에게 능동성 및 횡단관심사에 대한 모듈화 기능을 제공한다는 것을 지적하고 이들의 통합(converge)을 통해 차세대 반응형 미들웨어의 개발에 도움을 줄 것이라 주장하였다. 이외에도 EAOP[14,15] 연구 그룹에서 복합 이벤트와 유사한 기능을 AOP에서도 지원하도록 연구를 수행하고 있다. Douence 등은 [16]에서 AOP모델을 적용하여 전자상거래 시스템의 구현을 효과적으로 할 수 있음을 보였다. 하지만 아직까지 복합 이벤트 모델과 AOP의 본격적인 통합을 위해서는 해결되어야 할 과제들이 많이 있다. 두 모델을 모두 수용하는 이벤트 모델과 이벤트 연산자가 설계되어야 하며, 실제 응용들에 대해 이러한 통합 모델이 효과적임을 보여야 한다.

본 연구에서 제안하는 이벤트 모델은 Liu[11]와 실시간 모니터링 시스템[17,18]의 연구 결과에 기반한다. AOP와의 연동을 고려하여 이벤트 모델을 확장하며 이벤트 상관 기반의 네트워크 관리 시스템 응용에 적합한 이벤트 연산자를 설계한다.

3. 통합된 이벤트 모델

3.1 독립적인 이벤트 계층 지원 및 계층 관련 연산자

이벤트 명세(specification)시에 관련된 이벤트 그룹에 대해 동일한 처리를 요하는 경우가 있다. 만일 이러한 패턴이 응용에서 자주 나타난다면 그러한 그룹 관계를 이벤트 정의에 포함시켜 간략하게 명세할 수 있다.

기본 이벤트의 경우에는 시스템 이벤트로서 미리 정의된 계층이 있는 경우가 많으나[19], 복합 이벤트의 계층 정의에 관한 연구는 활발하지 않았다. 본 시스템에서 AOP에서 정의된 명명된 교차점(named pointcut)들은 자동으로 이벤트의 이름으로 인식된다.

AOP의 애스펙트(aspect)들 간에도 클래스처럼 상속이 가능하여 일견 애스펙트의 상속을 이용하여 이벤트들의 그룹관계를 나타낼 수 있는 것처럼 보인다. 하지만, 이벤트 그룹정보는 일반적인 프로그래밍 언어에서의 상속(inheritance)과는 다른 성격의 것이다. 프로그래밍 언어의 상속은 소프트웨어 재사용(reuse)을 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 우리는 응용 수준에서 복합 이벤트간의 그룹화를 효과적으로 제공해주는 독립적인 이벤트 계층 지원 방법을 제안한다. AOP에서의 교차점들은 모두 PointCutEvent 시스템 이벤트의 하위계층이 된다. 이것은 Java 언어에서 모든 클래스는 Object 클래스의 하위 클래스가 되는 것과 유사하다.

본 시스템에서 제공되는 이벤트 계층관련 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, CChildEvent subof CParentEvent), CParentEvent 이벤트는 CChildEvent을 포함하지만 CParentEvent! 이벤트는 CChildEvent을 포함하지 않게 된다.

3.2 복합 이벤트 정의를 위한 이벤트 연산자

OR(e_1, e_2)는 이벤트 e_1 혹은 e_2 중 어느 하나라도 검출된 경우에 발생한다. SEQ(e_1, e_2)는 이벤트 e_1, e_2 모두가 검출되었고 $@e_1 \leq @e_2$인 경우에 발생된다. AND(e_1, e_2)는 이벤트 e_1과 e_2 모두 검출되었을 때 발생한다. 위와 같이 정의된 순수한 AND 연산자는 e_1과 e_2의 타임스탬프와는 아무런 연관 관계가 없어서 실용적이지 못하다. 따라서, AND 연산자를 사용할 때 대개 다음과 같이 마감 시간(deadline)을 지정하여 두 이벤트의 시간차가 그 이내인 경우로 한정한다.

$$AND(e_1, e_2 \mid t_1) \equiv \text{true iff } AND(e_1, e_2) \wedge |@e_1 - @e_2| \leq t_1.$$

마감 시간 이외에도 지연 시간(delay)을 지정하여 두 이벤트의 시간차가 지정 시간 이상인 경우로 한정할 수도 있다. 다음의 예는 마감 시간과 지연 시간이 모두 지정된 경우이다.

$$AND(e_1, e_2 \mid t_1, t_2) \equiv \text{true iff } AND(e_1, e_2) \wedge t_2 \leq |@e_1 - @e_2| \leq t_1.$$

이벤트가 세개 이상 명시되었을 때는 일반적으로 다음과 같이 해석된다.

$$AND(e_1, e_2, \cdots, en \mid t_1, t_2) \equiv \text{true iff } (\forall i,j$$

$$AND(e_i, e_j) \wedge t_2 \leq |@e_i - @e_j| \text{where } i,j \in 1, 2, \cdots, n) \wedge (\max(@e_1, @e_2 \cdots, @e_n) - \min(@e_1, @e_2, \cdots, @e_n) \leq t_1).$$

이는 AND 연산자 뿐 아니라 다른 이벤트 연산자에도 적용된다.

ANY(e_1, e_2, ⋯, e_n, k) 이벤트는 e_1부터 e_n까지의 이벤트 중 최소 k개가 검출되었을 때 발생된다. COUNT (e, OP, OPRD, t) 연산자는 이벤트 e가 타임 윈도우 t 안에 발생된 횟수를 센 후 주어진 비교 연산자 OP와 피연산자 OPRD에 따라 발생된다. 예를 들어 COUNT (FrequentSamePurchase, ">=", 5, 10min)는 Frequent-SamePurchase 이벤트가 현재를 기준으로 10분 이내에 5번 이상 검출되었다면 발생된다. ~e는 이벤트 e의 검출이 되지 않았을 때 발생하는데 이 연산자는 시간 윈도우를 요구하는 다른 이벤트 연산자들과 함께 사용해야 한다. ~e가 시간 윈도우를 요구하는 연산자와 함께 쓰이면, 지정된 시간 윈도우내에 해당 이벤트 e가 검출되지 않을 때 발생한다.

- 연산자는 해당 이벤트를 배제하고자 할 때 사용한다. 예를 들어 EventA - EventB! or EventC 복합 이벤트는 EventA를 포함한 하위 이벤트들 혹은 EventC를 포함한 하위 이벤트에 대해 반응하지만 EventB에 대해서는 반응하지 않는다. 특히 EventB가 EventA의 하위 이벤트일 때 다음의 그림 1은 독립적인 이벤트 계층 중 사용자가 원하는 일부만을 이벤트 계층 연산을 통해 선택할 수 있음을 보인다.

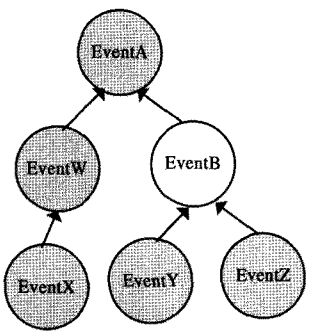


그림 1 이벤트 계층 지원

- 연산자와~연산자의 역할이 같은 것처럼 보이지만 그렇지 않다. 예를 들어 이벤트 B가 이벤트 A의 하위 계층 이벤트라 가정하자. 이벤트 인스턴스들이 A, B, C의 순서로 발생하면((A; C) ^ ~B)는 발생하지 않지만 (A-B ; C)는 발생된다. 이는 전자의 경우에는 A; C가 검출된 구간 내에 B가 없어야 하지만 후자의 경우에는 A-B 이벤트, 즉 B의 하위 계층이 아닌 A 이벤트가 C 이벤트와 SEQ 관계가 있으면 발생하기 때문이다.

3.3 이벤트 인스턴스 및 타이밍 제한 연산자

최근 실시간 시스템에서 발생되는 이벤트 인스턴스들 간의 정확한 타이밍 제약에 관한 많은 연구결과가 있었다[11,12,17,18]. 우리는 이러한 연구 결과를 본 연구에 적용하여 이벤트 명세 시에 시간적 관계를 쉽게 표현할 수 있도록 연산자를 확장한다.

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. 능동 데이터베이스에서는 이를 위해 소비 모드(consumption mode) 등을 정의하였으나, Liu[11]와 Mok[17] 등의 많은 후속 연구에서 이와 같은 소비 모드에 대한 문제점을 지적하였다.

우리는 모호성 없는 이벤트 명세를 위해 특정 이벤트 인스턴스를 지칭할 수 있는 인스턴스 지정 연산을 제공한다. 이벤트 명세 시에 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.

또한 현재 정의하고 있는 복합 이벤트 이외의 모든 이벤트 이름은 해당 복합 이벤트가 검출된 상태에서의 인스턴스를 가리킨다. 이 경우에도 이벤트 이름에 인덱스를 사용하여 예전의 인스턴스를 사용할 수 있다. 만일 인덱스를 생략하면 가장 최근에 검출된 인스턴스를 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.

다음의 그림 2는 어떤 시스템에서 두 종류의 이벤트가 발생한 연혁(event occurrence history)를 보여준다. 예를 들어 이벤트 e_1은 시스템 시간 0, 2, 4, 6에 검출되었다. 각각의 발생에 대해 이벤트 인스턴스가 발생된다. 앞에 제시된 표기법을 이용하면 @(e_1, 1) = 0, @(e_1, 2) = 2, @(e_1, 3) = 4, @(e_1, 4) = 6이다. R(e_2, @e_1, 3)은 e_1의 가장 최근 인스턴스의 타임스탬프를 기준으로 3번째 이전의 e_2 타입의 인스턴스를 가리킨다. 따라서 @R (e_2, @e_1, 3) = 2이다.

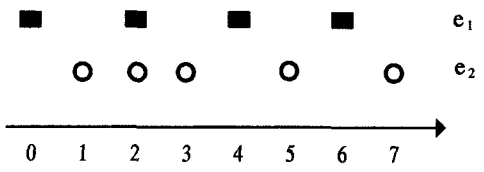


그림 2 이벤트 발생

4. 이벤트의 정의와 적용

4.1 기본 이벤트 정의

원시 이벤트(raw event)는 원시 이벤트 생성기(무선 센서, 이기종 간 어댑터, 네트워크 상태 탐지기 등)에서 이벤트에 해당하는 상황은 검출 되었으나 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 정의를 위한 문법 및 구체적인 한 예를 보여준다. 기본 이벤트 LightEvent는 LightSensor로 부터 발생되며 세개의 속성을 갖는다. 그 중 SerialNumber는 센서의 위치와 시간을 병합하여 만들고 나머지 속성은 해당되는 원시 이벤트의 속성을 가져온다. 노이즈 필터로써 LightFilter1 이후에 LightFilter2를 적용하고 있다.

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

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

4.2 복합 이벤트 정의

본 시스템에서의 이벤트는 다음과 같이 정의된다.

Cevent 복합 이벤트 이름 [**subof** 이벤트 이름들]{
  Pattern 이벤트 패턴절
  [**Where** 조건절]
  [**Action** 이벤트 처리절]
}

Cevent 키워드 뒤에는 정의하고자 하는 복합 이벤트의 이름을 지정한다. Pattern 키워드 뒤에는 검출하고자 하는 이벤트 패턴을 지정한다. Where 키워드 뒤의 조건절에는 부가적으로 이 복합 이벤트를 유발(trigger)시킬 조건을 지정한다. Action 키워드 뒤에는 이벤트 처리를 위한 메쏘드의 이름을 지정한다. 조건절과 이벤트 처리절은 생략 가능하다. 이때 이벤트 패턴절에 나오는 이벤트 인스턴스는 as를 사용하여 임의의 변수와 엮일(bind) 수 있다. 이렇게 엮인 변수는 해당 이벤트 인스턴스를 대표하며, 조건절 및 이벤트 처리절에서 해당 이벤트 인스턴스의 속성 등을 참고하기 위해 사용된다. 예를 들어 다음과 같은 복합 이벤트는 AbortTransaction 이벤트가 발생하면 그것의 속성인 critical의 값을 검사하여 참이면 Log.logmessage 메쏘드에 인자로 전달하여 호출한다.

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

컴퓨터 네트워크에서는 하나의 네트워크 부품 고장이 그 부품과 관계 있는 다른 장치나 다른 네트워크 부품의 동작에도 영향을 주어 다량의 관련 이벤트가 발생되기도 한다[11]. 이러한 경우에 특히 본 연구에서 제안하는 이벤트 계층 구조와 각종 이벤트 연산자가 관리자로 하여금 "이벤트 홍수"에 빠지지 않고 정확히 진단에 필요한 이벤트만을 필터링 할 수 있는 기능을 제공한다. 예를 들어 다음의 복합 이벤트 CascadingFailure는 가장 하위 계층의 부품 고장이 파급되어 2단계 및 1단계에 있는 노드들의 오작동이 감지된 경우에 발생된다.

```
Cevent CascadingFailure {
  Pattern ( SEQ( NodeFailureLevel3,
  NodeFailureLevel2, NodeFailureLevel1, 5min))
}
```

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 FrequentHalfOpen 이벤트를 발생시키고, 이러한 FrequentHalfOpen 이벤트가 짧은 시간 안에 반복되는 경우 DOS 공격에 대한 경보를 발생시킨다.

```
Cevent FrequentHalfOpen {
  Pattern ( SEQ(SynEvent as x, ~AckEvent,
  SynEvent as y, 5sec) )
```

```
  Where ( @this - @prev <= 10sec and
  x["source"] = y["source"] )
}

Cevent DosSymptom {
  Pattern ( COUNT(FrequentHalfOpen ,
  ">=", 10, 2min) )
  Action Call( Message.SendDOSWarning() )
}
```

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 SystemAllShutDown 이벤트는 제외한다.

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

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

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

본 연구의 응용은 네트워크 관리에만 제한되지 않고 네트워크 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.

```
Cevent FrequentSamePurchase {
  Pattern ( SEQ(Find as x, MoveToCart as y,
  Pay, 1min) )
  Where ( @this - @prev <= 1min and
  x["product_id"] = y["product_id"] )
}
Cevent FrequentSamePurchaseMoreThan5Within
10min {
  Pattern ( COUNT(FrequentSamePurchase , ">=",
  5, 10min) )
}
```

또 다른 예로써, 신용카드 회사에서는 다음과 같은 복합 이벤트를 이용하여 동일한 카드가 짧은 시간 안에 직전 거래 위치로부터 원거리에서 결제된다면 신용카드 사고를 사전 경계할 수 있다.

```
Cevent CreditCardFraudWarning {
  Pattern ( SEQ(CardUse as x, CardUse as y,
  10min) )
  Where (distance(x["location"], y["location"]) >
  100km and x["card_number"] = y["card_number"] )
  Action Call( Message.SendScamWarning() )
}
```

4.3 이벤트 모델의 표현력 평가

본 절에서는 이벤트 상관 기반의 네크워크 관리 시스템들의 이벤트 모델에 대한 표현력을비교한다. 능동 데이터베이스에서 처음으로 제안된 복합 이벤트는 기존에 단순한 통지(notify)의 목적으로 사용되던 이벤트에 조합성(composability)를 부여하였다. 여러 기본 이벤트의 조합으로 복잡한 상황(situation)에 대한 기술을 효과적으로 할 수 있는 복합 이벤트는 다양한 분야로 응용되었다. 특히 네트워크상에서 발생되는 여러 단편적인 증상(symptom)들의 연관 관계를 이용하여 실제 고장의 원인을 쉽게 파악할 수 있게 하는 장점으로, 복합 이벤트는 자연히 이벤트 상관 기반의 네트워크 관리 시스템으로 응용되었다[10,11].

표 1은 이벤트 상관 기반의 네트워크 관리 시스템 중 대표적인 시스템인 IMPACT[7], Hasan[10], SEL[20], JECTOR[11] 등과의 비교를 보인다. 본 연구에서 제안된 이벤트 모델이 특장점을 보이는 이벤트 계층 및 시간적 관계에 관련된 연산자를 중심으로 비교를 하고 있다.

IMPACT[7]는 복합 이벤트를 지원하지 않는 이벤트 상관 기반 네트워크 관리 시스템이다. 이벤트 계층 표현에 대한 지원은 일부 있으나 단순히 알람(alarm) 이벤트의 정적인 포함 관계 기술에 그치고 있다. 이후 능동 데이터베이스의 복합 이벤트의 개념이 등장하면서 Hasan[10] 등의 연구에서 본격적으로 복합 이벤트가 채용되기 시작하였다. 능동 데이터베이스에서 기원된 이벤트 모델들은 일반적으로 SEQ 및 고정된 시간 윈도우에 대한 지원을 제외하고는, 네트워크 관련 이벤트의 기술

표 1 이벤트 상관 기반의 네트워크 관리 시스템들의 표현력 비교

종류	IMPACT[7]	Hasan[10]	SEL[20]	JECTOR[11]	본시스템
기반 이벤트 모델	단순 이벤트 모델	복합이벤트 모델	복합 이벤트 모델	RTL	RTL
이벤트 명세시 이벤트타입에 의한 명세	지원	지원	지원	상위 응용에서 제공	지원
이벤트 명세시 인스턴스 수준의 명세	미지원	지원	제한적 지원	지원	지원
이벤트 계층 표현	제한적 지원	미지원	미지원	제한적 지원	지원
특정 이벤트 계층 단독지정 연산	미지원	미지원	미지원	미지원	지원
특정 이벤트 계층 제외 연산	미지원	미지원	미지원	미지원	지원
마감 시간 관계 표현	고정된 시간 윈도우만 지원	고정된 시간 윈도우만 지원	지원	지원	지원
지연 시간 관계 표현	미지원	미지원	지원	지원	지원
AOP와의 연동	미지원	미지원	미지원	미지원	지원

에 필요한 마감 시간(deadline) 혹은 지연 시간(delay) 등 시간적 관계 연산자의 지원이 부족하다. 또한, 같은 타입의 복수개의 이벤트 인스턴스들 중에서 특정한 인스턴스를 지정하는 연산자가 풍부하지 못하여 복잡한 네트워크 복합 이벤트를 기술하기에 적합하지 않다.

SEL[20]은 간결성을 중시한 복합 이벤트 모델이며 마감 시간 및 지연 시간 등에 대한 표현력을 증가시켰다. 하지만 이벤트 타입에 기반한 모델이기 때문에 인스턴스간의 복잡한 시간 관계를 나타내는 데에는 여전히 부족함이 있다.

Liu[11] 등이 개발한 JECTOR는 실시간 시스템의 시간적 양태(temporal behavior)를 기술하기 위해 개발된 Real-Time Logic(RTL)[12]을 확장한 복합 이벤트 모델에 기반하고 있다. 따라서 능동 데이터베이스의 이벤트 모델에 비해 이벤트들간의 시간적 제약에 대한 표현이 용이하고 특정 이벤트 인스턴스를 지칭할 수 있는 연산자가 풍부하다.

본 연구에서 제안된 이벤트 모델은 Liu[11] 등의 연구 결과를 개선하여 이벤트 계층 표현 및 최근 새로운 개발 방법으로 각광 받고 있는 AOP와의 이벤트 연동을 추가하였다. 특히 특정 이벤트 계층의 단독 지정 연산자(!)와 배제 연산자 (-)의 제공으로 이벤트 계층에 관련된 모델의 표현력이 강화되었다.

5. 결 론

본 연구에는 기존에 연구되었던 다양한 이벤트 모델들을 통합하여 네트워크 관리 시스템의 설계 및 구현에 적합하도록 확장하였다. 제안된 이벤트 모델은 AOP의 교차점과 이벤트 수준에서 연동될 수 있고 계층적 이벤트를 지원한다. 또한 이벤트 인스턴스를 지정할 수 있는 연산자를 제공하여 모델의 표현력을 증가시켰으며 시간 관계를 잘 표현 할 수 있는 연산자들을 추가하였다.

이와 같이 확장된 이벤트 모델로써 나타낼 수 있는 복잡한 시나리오와 그에 상응하는 복합 이벤트의 정의를 소개하였다. 마지막으로 기존의 이벤트 기반의 접근 방법들에 비해 본 연구에서 제안된 모델이 가지는 장점을 제시하였다. 제안된 이벤트 모델은 차세대 네트워크 관리 시스템을 위해 설계되었으나 이에 제한 되지 않고 다양한 응용에 사용될 수 있다.

향후 연구로는 제안된 이벤트 모델의 이벤트 검출 알고리즘 구현 및 실제 시스템에의 적용이 될 것이다. 제안된 이벤트 모델은 AOP의 교차점이 발생될 때마다 이벤트를 생성해야 하며 이벤트들간의 계층적 구조를 지원하여, 일반적인 복합 이벤트 시스템에 비해 수행 시간 부담이 예상된다. 또한 이벤트 인스턴스에 대한 연산자 지원 및 시간 관계 연산자 처리에 대한 고려도 되어야 한다. 정량적인 성능 측정 결과를 통해 증진된 표현력과 수행 시간 부담에 대한 분석을 할 수 있을 것이다.

또한 기존의 능동 데이터베이스에서 연구되었던 복합 이벤트의 여러 문제(issue)들을 AOP와 통합된 모델 하에서 다시 고려해 보는 것도 흥미로울 것이다. 예를 들어 Cilia[13] 등이 지적한 바와 같이 '즉시(immediate)', '지연(delayed)', '분리된 트랜잭션(detached)'을 지원하는 결합 모드(coupling mode)가 통합된 모델에서는 어떻게 적용될지 연구해 볼 수 있을 것이다.

참 고 문 헌

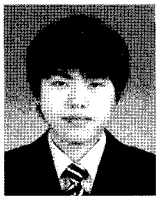
[1] D. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley, 2002.

[2] G. Kiczales et al., "Aspect-Oriented Programming," In *Proc. of the European Conference on Object-Oriented Programming (ECOOP)*, 1997.

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

[4] S. Chakravarthy et al., "Composite Events for Active Databases: Semantics, Contexts and Detec-

tion," In *Proc. of the International Conference on Very Large Data Bases (VLDB)*, 1994.
[5] U. Dayal et al., "The HiPAC Project: Combining Active Databases and Timing Constraints," *ACM SIGMOD RECORD*, **17**(1), 1988.
[6] N. H. Gehani and H. V. Jagadish, "Ode as an Active Database: Constraints and Triggers," In *Proc. of the International Conference on Very Large Data Bases (VLDB)*, 1991.
[7] G. Jakobson and M. Weissman, "Alarm Correlation," *IEEE Network*, **7**(6), 1993.
[8] K. Houck et al., "Towards a practical alarm correlation system," In *Proc. of the Symposium on Integrated network management*, 1995.
[9] S. Yemini et al., "High speed and robust event correlation," *IEEE Communications*, **34**(5), 1996.
[10] M. Hasan, "An Active Temporal Model for Network Management Databases," In *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management*, 1995.
[11] G. Liu et al., "Composite Events for Network Event Correlation," *Proc. of the IFIP/IEEE Symposium on Integrated Network Management*, 1999.
[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. Cilia et al., "The Convergence of AOP and Active Databases: Towards Reactive Middleware," In *Proc. of the International Conference on Generative Programming and Component Engineering*, 2003.
[14] EAOP Project. http://www.emn.fr/x-info/eaop/
[15] R. Douence and M. S¨udholt, "A Model and a Tool for Event-Based Aspect-Oriented Programming (EAOP)," *Technical Report 02/11/INFO*, Ecole des Mines de Nantes, 2002.
[16] R. Douence et al., "Sophisticated Crosscuts for E-Commerce," In *Proc. of the ECOOP Workshop on Advanced Separation of Concerns*, 2001.
[17] A. K. Mok et al., "Specifying Timing Constraints and Composite Events: An Application in the Design of Electronic Brokerages," *IEEE Transactions on Software Engineering*, 30(12), 2004.
[18] C. G. Lee et al., "Monitoring of Timing Constraints with Confidence Threshold Requirements," *IEEE Transactions on Computers*, 56(7), 2007.
[19] J. Carlson, "Event Pattern Detection for Embedded Systems," Malardalen University, Ph.D. Thesis, 2007.
[20] D. Zhu and A.S. Sethi, "SEL, a new event pattern specification language for Event Correlation," In *Proc. of the IEEE ICCN*, 2001.



이 기 성
2005년 성균관대학교 한문학과 학사. 2006년~2008년 (주)온네트 소프트웨어 엔지니어. 2009년~현재 중앙대학교 컴퓨터공학부 석사과정. 관심분야는 실시간 소프트웨어, 소프트웨어 개발 방법론

이 창 하
1995년 서울대학교 계산통계학과 학사 1997년 서울대학교 계산통계학과 석사 2005년 Univ. of Maryland at College Park 전산학과 박사. 2005년~2007년 미국 NCA Medical Simulation Center 연구원. 2007년~현재 중앙대학교 컴퓨터공학부 조교수. 관심분야는 실시간 렌더링, 가상현실 등

이 찬 근
1996년 중앙대학교 전자계산학과 학사 1998년 KAIST 전산학과 석사. 2005년 Univ. of Texas at Austin 전산학과 박사. 2005년~2007년 미국 인텔 소프트웨어 엔지니어. 2007년~현재 중앙대학교 컴퓨터공학부 조교수. 관심분야는 실시간 소프트웨어, 수행시간 모니터링, 소프트웨어 테스팅