1L-05

Event Calculusに基づく複合イベント処理について

金山 貴紀 † 石川 佳治 † 杉浦 健人 †
† 名古屋大学大学院情報科学研究科

1 はじめに

ストリーム的に得られるイベントからより高次のイベントを検出する**複合イベント処理**（complex event processing, CEP）は，近年大いに着目されている [3]．複合イベントを検出するアプローチの一つは，データストリームに対する問合せ言語に，正規表現などのパターン記述機能を持たせることである [2, 6]．

一方，人工知能の分野を中心に，イベントを論理的な枠組みで捉えようとするアプローチも研究されてきており，その代表的なものが **Event Calculus** である [5]．一階述語論理の一種である Event Calculus はイベントを論理的な枠組みで捉えられるものであり，複合イベントを柔軟に定義できる．加えて，Event Calculus を扱う推論器も公開されているため，その高度な推論機能を活用した複合イベント処理が考えられる．

本研究では Event Calculus に基づく複合イベント処理のフレームワークを示し，既存のデータストリーム管理システムと Event Calculus の推論器を用いた実装について述べる．

2 Event Calculus とは

Event Calculus にはさまざまな変種がある．ここでは文献 [1] で用いられているものを想定する．

Event Calculus の基本概念として event と fluent がある．**event** はある時点で発生したイベントを表す．たとえば $WakeUp(p)$ は，人物 p が起床するというイベントである．一方，**fluent** は時刻によって値が変わりうる性質を表す．たとえば $Awake(p)$ は人が起きているかどうかを表す fluent であり，$Awake(p) = \texttt{true}$ は人が起きているという状態を表す．fluent の値は，event の発生により更新されうる．

Event Calculus においては，いくつかの述語が提供されている．$Happens(e, t)$ は，event e が時刻 t において生じたことを示す．なお，Event Calculus は離散的な時間を扱っている．$HoldsAt(f, t)$ は，時刻 t における fluent f の値が `true` であることを示す．$Initiates(e, f, t)$ は，event e が時刻 t で発生したとき，fluent f の値が次の時刻から `true` になることを示す．逆に，$Terminates(e, f, t)$ は，event e が時刻 t で発生したとき，fluent f の値が次の時刻から `false` になることを示す．

On Complex Event Processing Based on Event Calculus
Atsuki Kanayama †, Yoshiharu Ishikawa †, Kento Sugiura †
† Graduate School of Information Science, Nagoya University

3 Event Calculus を用いた推論

Event Calculus は，シナリオと対象世界の公理を表現でき，この性質により推論に用いることができる．**シナリオ**は，具体的なイベントに対応する式である．**公理**（axiom）は，常に真と評価される式であり，推論を行う上で最も基本的な式である．公理により，既存のシナリオから別のシナリオを導くことができる．

たとえば，以下の式は人物 p が起床すると起きている状態になるという公理である．

$$Initiates(WakeUp(p), Awake(p), t) \tag{1}$$

人物 p が就寝することを表すイベント $FallAsleep(p)$ を用いると，人物 p が就寝すると起きている状態でなくなるという公理は以下のように書ける．

$$Terminates(FallAsleep(p), Awake(p), t) \tag{2}$$

$Nathan$ という人物が時刻 0 において起きている状態ではなく，時刻 1 において起床したというシナリオは，以下のように書ける．

$$\neg HoldsAt(Awake(Nathan), 0) \tag{3}$$

$$Happens(WakeUp(Nathan), 1) \tag{4}$$

式 (1),(2) の公理，式 (3),(4) のシナリオ，および Event Calculus の公理（どの推論でも用いる基本的な公理の集合）より，以下の式が導かれる．

$$HoldsAt(Awake(Nathan), 2) \tag{5}$$

これは，$Nathan$ が時刻 2 において起きていることを表す．このように，Event Calculus では対象世界の公理，シナリオ，Event Calculus の公理を用いて推論を実行できる．また，Event Calculus は一階述語論理の一種であるため，優れた表現能力を持ち，複合イベントを柔軟に定義できる．

Event Calculus を扱う推論器として，離散 Event Calculus 推論器 [4]（Discrete Event Calculus Reasoner）がある．この推論器の特徴は，複雑な Event Calculus の式を一度 SAT 問題に変換する点である．SAT 問題の効率的な解法は以前から盛んに研究されているため，複雑な推論も効率的に実行できる．本研究では，この推論器を用いて複合イベント処理を行う．

4 システムのアーキテクチャ

想定するシステムアーキテクチャを図 1 に示す．入力は **PE ストリーム**である，PE は複合イベントで

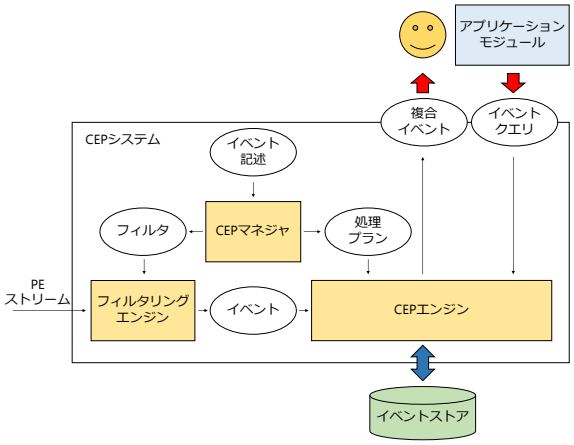

図 1: システムアーキテクチャ

ないプリミティブイベント（primitive event）を意味している．**イベント記述**（event description）は，システムに事前にユーザや管理者などにより与えられている複合イベントの導出のための公理の集合である．**CEP マネジャ**（CEP manager）は，与えられたイベント記述をもとに，**フィルタ**（filter）を導出する．これは，PE ストリームから必要なものだけを抽出するために利用される．

複合イベントの検出処理を担当するのが **CEP エンジン**（CEP engine）である．CEP マネジャから与えられた処理プランをもとに処理を行う．本研究では，CEP エンジンの一部に離散 Event Calculus 推論器を利用する．検出された複合イベントは，ユーザおよびアプリケーションモジュールに報告される．

5 複合イベント処理のアプローチ

本節では，Event Calculus に基づく複合イベント処理において，離散 Event Calculus 推論器を用いる方式について例を用いて説明する．基本的なアイデアは，複合イベントの定義を公理，ストリームからのイベントをシナリオとして推論器に入力し，複合イベント処理を実行することである．

以下の公理は，*SwitchOn*，*Plug* イベントによって，デバイス d の状態を表す fluent *SwitchedOn*，*PluggedIn* の値が変化することを示す．

$$Initiates(SwitchOn(d), SwitchedOn(d), t) \quad (6)$$

$$Initiates(Plug(d), PluggedIn(d), t) \quad (7)$$

以下の公理は，デバイス d が稼働状態になるのは，スイッチが押された状態でかつ，電源プラグが差し込まれた状態であるということを示す．

$$HoldsAt(On(d), t) \Leftrightarrow HoldsAt(SwitchedOn(d), t) \wedge HoldsAt(PluggedIn(d), t) \quad (8)$$

式 (6),(7),(8) をイベント記述に含める．*device1* というデバイスに関する *Plug* イベント，*SwitchOn* イベントが順に発生し，システムに入力されると推論器には以下の式がシナリオとして入力される．

$$Happens(Plug(device1), 0) \quad (9)$$

$$Happens(SwitchOn(device1), 1) \quad (10)$$

推論器は，公理 (6),(7),(8)，シナリオ (9),(10) および Event Calculus の公理より，時刻 2 においてデバイス *device1* が稼動状態となることを示す以下の式を推論し出力する．

$$HoldsAt(On(device1), 2)$$

また，プリミティブイベントに対応するイベントを，逐次イベントストアに追加する．推論器では，これまでに検出されていない，差分となる複合イベントのみを検出し，報告する．この際，わずかな追加のみに対して，一から推論を行うのはオーバヘッドが高いため，本研究では新たなプリミティブイベントに影響する公理を選択して部分的な推論を行う．これにより，差分の計算を効率化する．

6 まとめと今後の課題

本稿では Event Calculus に基づく複合イベント処理を実現するアプローチについて述べた．今後は，システムとして提供する機能について具体化を進め，実装方式に関する開発を行う．具体的には，ストリーム的な処理だけでなく，蓄積されたイベントストアに対して，その場ごとのアドホックな問合せを行う機能の実現が考えられる．

謝辞

本研究は，JST COI（Center of Innovation）プログラムおよび科研費（16H01722, 26540043）による．

参考文献

[1] *Commonsense reasoning*. Morgan Kaufmann.

[2] H.-L. Bui. Survey and comparison of event query languages using practical examples. Graduate Thesis, Institut für Informatic, Ludwig-maximilians-Universität München, Mar. 2009.

[3] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3), 2012.

[4] Commonsense reasoning with the discrete event calculus reasoner. http://decreasoner.sourceforge.net/.

[5] Wikipedia: Event calculus. http://en.wikipedia.org/wiki/Event_calculus.

[6] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *ACM SIGMOD*, pp. 407–418, 2006.