D-019

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して，正規表現などのパターン記述機能を持つ問合せ言語を使用することである [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，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** は状態などを表し，各時刻において `true` または `false` の値をとる．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．

3 Event Calculus を用いた推論

Event Calculus は，対象世界の公理とシナリオを表現でき，この性質により推論に用いることができる．**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．**シナリオ**は，具体的なイベントに対応する式である．公理により，既存のシナリオから別のシナリオを導出できる．

たとえば，式 (1) 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．

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

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，式 (2),(3) のように書ける．

$$\neg HoldsAt(Awake(Nathan), 0) \tag{2}$$

$$Happens(WakeUp(Nathan), 1) \tag{3}$$

式 (1) の公理，式 (2),(3) のシナリオより，推論結果として式 (4) のシナリオが導かれる．

$$HoldsAt(Awake(Nathan), 2) \tag{4}$$

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とシナリオを用いて推論を実行できる．なお，推論にはこれらに加えて，基本的な公理の集合である Event Calculus の公理が暗黙に使用される．

Event Calculus を扱うシステムとして，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 は複合イベントでない**プリミティブイベント**（primitive event）を意味する．**イベント記述**（event description）は，ユーザや管理者などにより，事前にシステムに与えられている複合イベントを導出する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．**CEP エンジン**（CEP engine）は，CEP マネジャから与えられた処理プランをもとに，複合イベントの検出処理を行う．

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

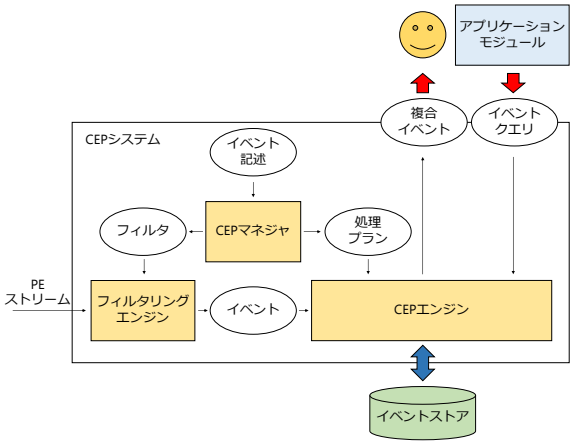

図 1: システムアーキテクチャ

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

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，推論結果の一部を検出した複合イベントとして出力することである．

オンラインストアの例を考える．顧客 c が商品 i を注文すると，注文 ID が o の注文が処理される．各注文について，配送，支払いが完了すると，それぞれ配送済み状態，支払い済み状態となる．

$$Initiates(Order(o,c,i), Ordered(o,c,i), t) \quad (5)$$

$$HoldsAt(Ordered(o,c,i),t) \Rightarrow Initiates(Delivery(o), Delivered(o), t) \quad (6)$$

$$HoldsAt(Ordered(o,c,i),t) \Rightarrow Initiates(Payment(o), Paid(o), t) \quad (7)$$

注文後に，配送と支払いの両方が完了すると，取引成功状態となる．

$$HoldsAt(Delivered(o),t) \Rightarrow Initiates(Payment(o), Succeeded(o), t) \quad (8)$$

$$HoldsAt(Paid(o),t) \Rightarrow Initiates(Delivery(o), Succeeded(o), t) \quad (9)$$

式 (5),(6),(7),(8),(9) をイベント記述に加える．*Alice* が商品 *cards* を注文し，この注文の注文 ID が 54 であったとする．*Alice* が支払いを終え，商品の配送が完了すると，以下のイベントが順にシステムに入力される．

$$Happens(Order(54, Alice, cards), 0) \quad (10)$$

$$Happens(Payment(54), 1) \quad (11)$$

$$Happens(Delivery(54), 2) \quad (12)$$

推論器は，公理 (5),(6),(7),(8),(9) およびシナリオ (10),(11),(12) より，式 (13) を出力する．

$$HoldsAt(Succeeded(54), 3) \quad (13)$$

式 (13) は，時刻 3 においてこの取引が成功したことを示す．このようにして，推論器による複合イベントの検出ができる．

また，プリミティブイベントを逐次イベントストアに追加する．これにより，問合せによって過去の状態を知ることができる．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.