RDBと連携したイベント計算による複合イベント処理

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　〒 464–8601 愛知県名古屋市千種区不老町
†† 名古屋大学大学院情報学研究科　〒 464–8601 愛知県名古屋市千種区不老町
E-mail: † {kanayama, sugiura}@db.ss.is.nagoya-u.ac.jp, ishikawa@i.nagoya-u.ac.jp

あらまし　近年，IoT やビッグデータに関連し，ストリーム的に得られるデータをリアルタイムに処理することが求められるようになった．複合イベント処理はこのようなストリーム的に得られるイベントと呼ばれるデータを処理する技術である．複合イベント処理において，単純なイベントから，それらを組み合わせた複合イベントを検出する手法が課題となっている．本研究では，RDB による問合せ処理能力と人工知能の分野で利用されているイベント計算の技術を組み合わせ，意味的な複合イベント処理の手法とそのフレームワークを提案する．

キーワード　イベント計算，複合イベント処理

1. はじめに

1.1 背　景

近年，情報通信技術（information and communication technology, ICT）の発達により，Web 上や実世界のデータなど，様々なデータが利用可能になってきている．これに従い，日々大量に生成されるデータをどのように有効活用するのかが課題となり，IoT やビッグデータが注目されてきている．特に素早い対応や意思決定が必要な分野では，リアルタイムに得られるタイムスタンプ付きのデータ（データストリーム）を処理する要求が高まっており，**データストリーム処理**（data stream processing）が以前から盛んに研究されている [1].

データストリーム処理技術の一つとして**複合イベント処理**（complex event processing, CEP）がある．複合イベント処理は，データストリームから得られる単純なイベントを監視し，そこから意味のあるイベントを検出する技術であり，近年大いに着目されている [3]．単純イベントとそれらから導かれる意味のある複合イベントがあれば，幅広い分野で利用することができる．例として，道路のトラフィックから渋滞を検出する，株式市場において株価の変動から特定のパターンを検出する，オンラインショッピングにおいて注文情報からクレジットカードの不正利用を検出する，といったことが考えられる．これらの意味のある複合イベントをリアルタイムに検出できれば，信号機の点灯タイミングを変更して渋滞を解消したり，株式取引において素早い意思決定を行ったり，オンラインショッピングにおいて脅威に迅速に対応し安全性を高めたりすることが可能となる．

本研究ではこのような複合イベントの定義に**イベント計算** [5,8]（event calculus）に基づく記述を用いる．イベント計算はイベントを論理的な枠組みで捉えるアプローチであり，人工知能の分野を中心に研究されている．イベント計算の記述は一階述語論理の一種であり，基本概念としてイベントに加えてフルーエントと呼ばれるものがある．フルーエントは実世界において時間変化する状態を表す特殊な述語記号であり，イベントの発生に伴いその値が変化する．イベント計算の記述では各イベントがどのような状態変化を引き起こすのかが明確に定義されるため，意味的な複合イベント処理が可能となる．

例として，病院において RF タグを取り付けた機器の位置情報を監視し，その位置と誰が使用しているかという稼働状況を管理する例を考える．ここで，データストリームから以下の 2 種類のイベントが与えられるとする．

- 移動イベント $Move(d,l)$: 機器 d が位置 l に移動したことを表す．
- 使用開始イベント $Borrow(s,d)$: 職員 s が機器 d を使用し始めたことを表す．

これらのイベントから，誰がどの機器をどこで使用しているのかを把握したい．そこで，機器の位置及び使用状況を表す以下のフルーエントを定義する．

- 位置情報 $Located(d,l)$: 機器 d が位置 l にあることを表す．
- 使用状況 $Using(s,d)$: 職員 s が機器 d を使用していることを表す．

つまり，イベント $Move(d,l)$ の生起によってフルーエント $Located(d,l)$ の，イベント $Borrow(s,d)$ の生起によってフルーエント $Using(s,d)$ の真偽値が変化する．このとき，職員 s が機器 d を位置 l で使用していること（$UsingAt(s,d,l)$）は，以下のような一階述語論理で記述できる．

$$UsingAt(s,d,l) \Leftrightarrow Using(s,d) \wedge Located(d,l)$$

しかし，この記述では時間変化する実世界の状態は表せない．そこで，イベント計算では各時刻 t におけるフルーエント f の真偽値を述語 $\texttt{HoldsAt}(f,t)$ を用いて以下のように表している．

$$\texttt{HoldsAt}(UsingAt(s,d,l),t)$$
$$\Leftrightarrow \texttt{HoldsAt}(Using(s,d),t) \wedge \texttt{HoldsAt}(Located(d,l),t)$$

HoldsAt(f,t) を用いない場合と異なり，ある時刻 t における機器の稼働状況を記述できている．このように，イベントによって変化する状態を考えることで，より複合イベントの意味を重視した記述が可能となるのがイベント計算を用いる利点の一つである．

1.2 課　　題

イベント計算を用いた知識表現・推論の技術について解説した [7] の著者は，イベント計算の推論器の実装として Discrete Event Calculus Reasoner [4] を開発している．この推論器では，対象とする世界のルールである公理と具体的なイベントである既知のシナリオから，未知のシナリオを推論によって導出できる．そのため，複合イベントの検出に利用できる．

しかし，この推論器ではイベント計算の式を一度 SAT 問題に変換して処理を行うため [9]，スケーラビリティが無いという問題がある．スケールの大きい問題では，リアルタイムに複合イベントを検出することが困難である．

1.3 提　　案

本研究では，イベント計算と RDB を組み合わせた複合イベント処理のフレームワークを提案する．複合イベントの定義にイベント計算の記述を用いることで，意味的な複合イベント処理を実現する．推論器によって，イベントの発生の有無とフルーエントの値を決定する際には，問題のドメイン全体を SAT 問題に変換する．しかし，イベントの発生とフルーエントの値の変化を逐次 RDB に記録し，これらの履歴データを使うことで，一部の推論を省略できる．また，複合イベント処理のコンテキストでは，単位時刻に生起するイベントの数を制限したり，ある時刻におけるフルーエントの値がすべて確定していると仮定したりできる．これにより，実際に解く問題のサイズを小さくすることで，RDB の問合せ処理によって推論を可能にし，効率化を実現する．さらに，一部の条件を満たす推論について，実体化ビューをインクリメンタルに管理する手法 [2] を利用し，新たに発生したイベントによる差分のみから推論を行う．

1.4 構　　成

本稿の構成は以下の通りである．まず，2. でイベント計算について述べる．3. で提案するフレームワークについて述べる．4. で提案手法について述べる．最後に，5. で本稿のまとめと今後の課題について述べる．

2. イベント計算

本節では，イベント計算の概要について紹介する．イベント計算は，等号を持つ多ソートの述語論理に基づいている．イベント，フルーエント，時刻，ドメインオブジェクトに対するソートが存在する．イベント計算の古典論理の公理化は，まず Shanahan [6] によって示された．なお，イベント計算にはさまざまな変種があるが，ここでは文献 [7] で用いられている離散的なイベント計算を想定する．また，本章で紹介するイベント計算独自の述語を表 1 にまとめる．

2.1 イベントとフルーエント

イベント計算の基本概念として**イベント**と**フルーエント**がある．

表 1: イベント計算における述語

述語	意味
Happens(e,t)	イベント e が時刻 t で発生する
HoldsAt(e,t)	フルーエント f の値が時刻 t で true である
Initiates(e,f,t)	イベント e が時刻 t で発生したとき，フルーエント f の値が時刻 $t+1$ で true に変化する
Terminates(e,f,t)	イベント e が時刻 t で発生したとき，フルーエント f の値が時刻 $t+1$ で false に変化する

イベントはある時点で発生した事象を表す述語記号であり，式 (1) のように記述する．なお，E はイベント名，$a_1,\ldots,a_n$ は引数であり，各引数の型は整数または文字列とする．

$$E(a_1,\ldots,a_n) \tag{1}$$

例えば，1. で述べた例で言えば，機器の移動を表す $Move(d,l)$ と機器の使用を表す $Borrow(s,d)$ がイベントである．

一方，**フルーエント**は実世界の状態を表す述語記号であり，式 (2) のように記述する．なお，F はフルーエント名，$b_1,\ldots,b_m$ は引数であり，各引数の型は整数または文字列とする．

$$F(b_1,\ldots,b_m) \tag{2}$$

例えば，1. で述べた例で言えば，機器の位置を表す $Located(d,l)$，機器の使用状況を表す $Using(s,d)$，及び機器の使用者と位置の情報を表す $UsingAt(s,d,l)$ がフルーエントである．

イベント計算において，イベントの生起状況やフルーエントの真偽値は時間の経過とともに変化するため，専用の述語 Happens(e,t) 及び HoldsAt(f,t) を用いて表す．各述語とその真偽値が表す意味は以下のとおりである．

- Happens(e,t)：イベント e が時刻 t に生起している．
- $\neg$ Happens(e,t)：イベント e が時刻 t に生起していない．
- HoldsAt(f,t)：フルーエント f の真偽値が時刻 t において true である．
- $\neg$ HoldsAt(f,t)：フルーエント f の真偽値が時刻 t において false である．

例えば，ある機器 *D1* が場所 *HospitalRoom1* を経由して，場所 *OperationRoom1* へ移動したとする．このとき，*HospitalRoom1* に到着した時刻が 1，*OperationRoom1* に到着した時刻が 2 であるとき，各時刻の移動はイベント *Move* を用いて以下のように表せる．

$$\texttt{Happens}(Move(D1, HospitalRoom1), 1)$$
$$\texttt{Happens}(Move(D1, OperationRoom1), 2)$$

2.2 公理によるルール記述

公理は常に値が `true` となる式である．イベント計算では，これを用いて対象世界のルールを記述できる．複合イベント処理においては，複合イベントを定義するのに利用できる．

まず，公理に関する説明の準備として，イベントの生起がフルーエントの真偽値に与える影響を表す述語記号を以下に示す．

- $\texttt{Initiates}(e, f, t)$：イベント e が時刻 t で発生したとき，フルーエント f の値が時刻 $t+1$ で `true` に変化する．
- $\texttt{Terminates}(e, f, t)$：イベント e が時刻 t で発生したとき，フルーエント f の値が時刻 $t+1$ で `false` に変化する．

同様に，イベントの生起やフルーエントの変化に関する条件の記述方法を示す．条件 γ は以下の項目に従って帰納的に定義する．項（term）は変数または定数とする．

- τ_1 および τ_2 が項であるとき，項の比較（$\tau_1 < \tau_2$，$\tau_1 \leqq \tau_2$，$\tau_1 = \tau_2$，$\tau_1 \geqq \tau_2$，$\tau_1 > \tau_2$，$\tau_1 \neq \tau_2$）は条件である．
- f がフルーエントであるとき，$\texttt{HoldsAt}(f, t)$ と $\neg\texttt{HoldsAt}(f, t)$ は条件である．
- γ_1 および γ_2 が条件であるとき，$\gamma_1 \wedge \gamma_2$ および $\gamma_1 \vee \gamma_2$ は条件である．
- v が変数かつ γ が条件であるとき，v に関する存在限量子 $\exists v\, \gamma$ は条件である．
- 上記以外は条件ではない．

本稿では，公理の記述として，文献 [7] で述べられているもののうち以下の 4 つの形式を使用する．なお，以下の式において，π は `Initiates` 若しくは `Terminates` のいずれかを示す．

a） 影 響 式

イベントの影響によりフルーエントの真偽値が変化することを表し，以下の式で記述する．

$$\pi(e, f, t) \tag{3}$$

例えば，機器 d が場所 l に移動した（イベント $Move(d, l)$ が生起した）とき，機器 d の所在が場所 l になる（フルーエント $Located(d, l)$ が `true` になる）ことは以下のように記述する．

$$\texttt{Initiates}(Move(d, l), Located(d, l), t)$$

b） 影 響 公 理

イベントの及ぼす影響がそのイベントの発生したコンテキスト（条件 γ）に依存することを表し，以下の式で記述する．

$$\gamma \Rightarrow \pi(e, f, t) \tag{4}$$

例えば，機器 d が場所 l_1 に移動した（イベント $Move(d, l_1)$ が生起した）とき，機器 d が l_1 以外の場所 l_2 には存在しない（フルーエント $Located(d, l_2)$ が `false` になる）ことは以下のように記述する．

$$l_1 \neq l_2 \Rightarrow \texttt{Terminates}(Move(d, l_1), Located(d, l_2), t)$$

c） トリガー公理

特定の条件が成立したときにイベントが発生することを表し，以下の式で記述する．

$$\gamma \Rightarrow \texttt{Happens}(e, t) \tag{5}$$

例えば，機器 d が部屋 *HospitalRoom1* にある（フルーエント $Located(d, HospitalRoom1)$ が `true` の）とき，警告を発する（イベント $Warn(d, HospitalRoom1)$ を生起させる）ことは以下のように記述する．

$$\texttt{HoldsAt}(Located(d, HospitalRoom1), t) \Rightarrow \texttt{Happens}(Warn(d, HospitalRoom1), t)$$

d） 状 態 制 約

常に成立する条件や，フルーエントの値と条件の関係を表し，以下の式で記述する．

$$\gamma \tag{6}$$

$$\texttt{HoldsAt}(f, t) \Leftrightarrow \gamma \tag{7}$$

例えば，1. の例で示したように，ある機器 d の使用状況（フルーエント $Using(s, d)$）と位置（フルーエント $Located(d, l)$）から，職員 s が機器 d を場所 l で使用していること（フルーエント $UsingAt(s, d, l)$）を導く式は以下のようになる．

$$\texttt{HoldsAt}(UsingAt(s, d, l), t) \Leftrightarrow \texttt{HoldsAt}(Using(s, d), t) \wedge \texttt{HoldsAt}(Located(d, l), t)$$

2.3 イベント計算における推論

イベント計算では，前述したルール記述に用いる公理以外に，時間経過に伴うフルーエントの真偽値の変化を表す**イベント計算の公理**が定義されている．イベント計算の公理において重要となる概念が，**常識的慣性の法則**である．あるフルーエントが慣性の法則に従うとき，そのフルーエントの真偽値は時間変化しない．つまり，何らかのイベントが生起し，かつそのイベントの与える影響が影響式若しくは影響公理で定義されているときのみ，フルーエントの真偽値が変化する．一方，フルーエントが慣性の法則から解放されているとき，フルーエントの真偽値は推論されるまで定まらない．つまり，状態制約（式 (7)）によって，他のフルーエントの真偽値からその値を推論する．なお，本来のイベント計算ではフルーエントの慣性の法則への従属状態を変化させられるが，本稿ではその処理は行わない．つまり，影響式若しくは影響公理によって真偽値の変化するフルーエントは常に慣性の法則に従い，状態制約によって真偽値を推論されるフルーエントは常に慣性の法則から解放されているとする．

例えば，移動に伴う機器の位置の変化を表す以下の影響式が定義されており，かつ初期状態（$t = 0$）で機器 *D1* が部屋 *HospitalRoom1* に置いてあったとする．

$$\texttt{Initiates}(Move(d, l), Located(d, l), t)$$

$$\texttt{HoldsAt}(Located(D1, HospitalRoom1), 0)$$

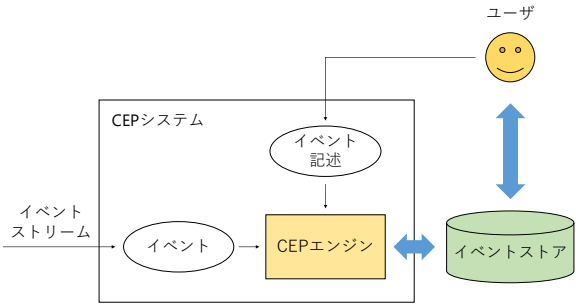

図 1: システムアーキテクチャ

ここで，ある機器 $D1$ が手術室 $OperationRoom1$ に移動したとき，その移動は以下のイベントで表せる．

$$\texttt{Happens}(Move(D1, OperationRoom1), 0)$$

上述のとおりイベント $Move(d,l)$ の与える影響式が定義されているため，イベント計算の公理に従い，次の時刻におけるフルーエント $Located(d,l)$ の真偽値は変化する．

$$\texttt{HoldsAt}(Located(D1, OperationRoom1), 1)$$

更に，フルーエント $Located(d,l)$ は慣性の法則に従っているため，時刻 $t=1$ でイベント $Move(d,l)$ が生起しなければ時刻 $t=2$ でもその真偽値は同じである．

$$\texttt{HoldsAt}(Located(D1, OperationRoom1), 2)$$

以上が，イベント計算における推論処理の基本的な流れである．

3. 提案フレームワーク

本節では，本研究で提案するシステムのフレームワークを述べる．まず，システムのアーキテクチャについて述べ，イベントとフルーエントの RDB における管理方法を説明する．次に，システムの使用例を挙げながら処理の流れを説明する．

3.1 システムアーキテクチャ

想定するシステムアーキテクチャを図 1 に示す．入力は**イベントストリーム**であり，ユーザによって事前定義された**イベント記述**に基づき **CEP エンジン**が新たなイベントの発生やフルーエントの値の変化を計算する．イベント及びフルーエントは全て**イベントストア**に記録し，ユーザによるアドホックな問合せを可能とする．なお，本論文ではイベント計算に基づく複合イベントの検出及び記録までを対象とし，検出した複合イベントの通知については扱わない．つまり，推論により導かれた複合イベントをテーブルに挿入するまでの処理についてのみ述べる．

以下，システムの各構成要素について説明する．

a） イベントストリーム

イベントストリームはイベントのデータストリームであり，生起したイベントのインスタンス $E(A_1,\ldots,A_n)$ が順にシステムに入力される．なお，本論文において時刻はイベントの到着順と同義であり，システムに入力された際にシステムのもつ現在時刻と同一の値が割り当てられる．例えば，システムの現在時刻が t のときに 2 つのイベント $E(A_1,\ldots,A_n)$ 及び $E(B_1,\ldots,B_n)$ が入力された場合，まず $\texttt{Happens}(E(A_1,\ldots,A_n),t)$ として処理される．その後，システムの現在時刻が $t+1$ となり，残りのイベント $\texttt{Happens}(E(B_1,\ldots,B_n),t+1)$ が処理される．ただし，入力されたイベントを処理した際にトリガー公理によって新たなイベントが生起した場合，次に入力されるイベントよりも優先して処理する．つまり，上述の例で $\texttt{Happens}(E(A_1,\ldots,A_n),t)$ を処理した際にイベント $E(C_1,\ldots,C_n)$ がトリガー公理により導かれたとき，まず $\texttt{Happens}(E(C_1,\ldots,C_n),t+1)$ として処理する．その後，残りのイベント $\texttt{Happens}(E(B_1,\ldots,B_n),t+2)$ を処理する．

b） イベント記述

イベント記述は対象とする世界に関する記述であり，イベント記述に含まれる式は常に `true` と評価される．つまり，対象とする世界において常に満たされるべき公理として記述する．複合イベント処理においては複合イベントの定義やイベント・フルーエント間の関係を定義する手段である．ユーザが予めシステムにイベント記述を登録することで，システムは登録されたイベント記述に基いて複合イベント処理を行う．なお，イベント記述として登録できる式には，複合イベント処理のユースケースに合わせた制限がある．許可される記述には，2. で述べた 4 種の公理，時刻 0 において `true` であるフルーエントを指定する記述，キー制約に関係する記述がある．

c） CEP エンジン

CEP エンジンは，システムに登録されたイベント記述をもとに推論処理を行い，複合イベントの検出を行う．ただし，離散的なイベント計算 [4, 7] において推論処理は SAT 問題へと変換されるが，本論文では RDB に基づく推論処理を行う．つまり，イベント記述から適切な SQL を生成し，イベントストア（RDB）に対して問合せを実行することで，推論結果となるテーブルないしタプルを生成する．

d） イベントストア

イベントストアは RDB であり，発生したイベントとフルーエントの値の時間変化を逐次記録する．各イベント・フルーエントの型に基づき，その履歴を記録するテーブルと現在時刻に関するもののみを記録するテーブルを自動生成する．このテーブルの自動生成については次節で詳しく述べる．

3.2 イベントとフルーエントの管理

イベント・フルーエントは RDB のテーブルで管理する．その管理方法について述べる．イベント・フルーエントはともに時刻の情報を持った履歴テーブルと，現在時刻に関するテーブルの 2 種類で管理する．これらのテーブルは，イベント・フルーエントの種類ごとに作成される．

3.2.1 イベントの管理

入力されたイベントは，逐次 RDB に保存する．イベント $E(a_1,\ldots,a_n)$ を定義すると，対応するテーブル $E(a_1,\ldots,a_n,t)$ が自動的に作成されるものとする．テーブルの列 $a_1,\ldots,a_n$ はイベントの引数 $a_1,\ldots,a_n$ にそれぞれ対応しており，各変数の

型は一致するものとする．t はイベントが発生した時刻を表す．ここで，以下の 2 つは等価である．

- テーブル E にタプル $(a_1, \ldots, a_n, t)$ が存在する
- 時刻 t において，イベント $E(a_1, \ldots, a_n)$ が発生した

例えば，$Move(D1, HospitalRoom1)$ 及び $Move(D1, OperatingRoom1)$ がこの順で入力され，それぞれ時刻 $t = 0$ と $t = 1$ のイベントとして処理されたとき，イベント $Move(d, l)$ の生起を表すテーブルは表 2 のようになる．

表 2: `Happens`$(Move(d, l), t)$ テーブル

d	l	t
D1	*HospitalRoom1*	0
D1	*OperatingRoom1*	1

また，現在時刻において発生したイベントのみを保持するテーブル $E_C(a_1, \ldots, a_n, t)$ も自動的に作成される．このテーブルについては，以下の 2 つは等価である．

- テーブル E_C にタプル $(a_1, \ldots, a_m)$ が存在する
- 現在時刻において，イベント $E(b_1, \ldots, b_m)$ が発生した

例えば，表 2 と同じ状況において現在時刻が $t = 1$ であるとき，対応するテーブルは表 3 のようになる．

表 3: 現在時刻におけるイベント $Move(d, l)$ の生起を表すテーブル

d	l
D1	*OperatingRoom1*

3.2.2 フルーエントの管理

フルーエントの値が `true` であった期間はテーブルで管理する．フルーエント $F(b_1, \ldots, b_m)$ を定義すると，対応するテーブル $F(b_1, \ldots, b_m, t_s, t_e)$ が自動的に作成されるものとする．テーブルの列 $b_1, \ldots, b_m$ はフルーエントの引数 $b_1, \ldots, b_m$ にそれぞれ対応しており，各変数の型は一致するものとする．t_s, t_e はそれぞれフルーエントの値が `true` である時区間の開始時刻と終了時刻を表す．ここで，以下の 2 つは等価である．

- テーブル F にタプル $(b_1, \ldots, b_m, t_s, t_e)$ が存在する
- 時区間 $[t_s, t_e)$ において，フルーエント $F(b_1, \ldots, b_m)$ の値が `true` である

ただし，タプルの t_e の値が NULL の場合は対応する時区間は $[t_s, t_n]$ とする．ここで，t_n はシステムがもつ現在時刻を表す．例えば，表 2 のようにイベント $Move(d, l)$ が生起したとき，対応する $Located(d, l)$ テーブルは表 4 である．

また，現在の値が `true` であるフルーエントを保持するテーブル $F_c(b_1, \ldots, b_m)$ も自動的に作成される．このテーブルについては，以下の 2 つは等価である．

表 4: `HoldsAt`$(Located(d, l), t)$ テーブル

d	l	ts	te
D1	*HospitalRoom1*	0	1
D1	*OperatingRoom1*	1	NULL

- テーブル F_C にタプル $(b_1, \ldots, b_m)$ が存在する
- 現在時刻 t_n において，フルーエント $F(b_1, \ldots, b_m)$ の値が `true` である

イベントの場合（表 3）と同様に，現在時刻が $t = 1$ のとき表 4 に対応するテーブルは表 5 である．

表 5: 現在時刻における $Located(d, l)$ の状態を表すテーブル

d	l
D1	*OperatingRoom1*

3.3 処理の流れ

本節では，提案システムにおける処理の大まかな流れについて説明する．提案システムでは，各時刻 t において以下の手順を繰り返す．

（1） 状態制約を評価し各フルーエントの真偽値を求め，RDB のテーブルに反映する．

（2） トリガー公理を評価し，生起するイベントを決定する．

（3） データストリームからの入力イベントまたはトリガー公理により生起したイベントに対し影響式・影響公理を評価し，次の時刻において値が変化するフルーエントを求める．

（4） 時刻を 1 だけ進める（$t \leftarrow t + 1$）．

（5） 手順 3 において求めたフルーエントの真偽値をテーブルに反映する．

以下，各手順について具体的な例を用いて説明する．

3.4 処理の具体例

3.4.1 想定シナリオ

病院には多くの医療機器が存在し，これらを医療スタッフが使用している．これらの機器の位置と稼働状況をリアルタイムに把握したいという要求がある．機器を適切に管理し，現在誰がどこでその機器を使用しているのかを瞬時に知ることができれば，機器の使用，設置，移動の際の手続きが簡単になるというメリットもある．そこで，提案するシステムによって医療機器を管理するという状況を想定する．なお，各機器の位置情報と機器 ID は貼付された RF タグによって，スタッフの職員 ID は職員証のバーコードをバーコードリーダーで読み込むことによって得られるとする．

このシナリオにおいて，存在するイベントとフルーエントは以下のとおりである．

a） イ ベ ン ト

- $Move(d, l)$：機器 d が場所 l に移動する
- $Borrow(s, d)$：スタッフ s が機器 d を借りる

b） フルーエント

- $Located(d,l)$：機器 d が場所 l に存在する
- $Using(s,d)$：スタッフ s が機器 d を使用中である
- $UsingAt(s,d,l)$：スタッフ s が機器 d を場所 l で使用中である

3.4.2 イベント記述の登録

想定シナリオにおけるイベント及びフルーエントの関係をイベント記述として次のように定義する．なお，このシナリオにおいてトリガー公理は存在しない．

a） 影響式・影響公理

まず，イベント $Move(d,l)$ の与える影響を式 (8) 及び式 (9) のように記述する．

$$\texttt{Initiates}(Move(d,l), Located(d,l), t) \tag{8}$$

$$\texttt{HoldsAt}(Located(d,l_1),t) \wedge l_1 \neq l_2 \Rightarrow \texttt{Terminates}(Move(d,l_2), Located(d,l_1), t) \tag{9}$$

式 (8) は影響式であり，機器 d が場所 l に移動する（イベント $Move(d,l)$ が発生する）と，機器 d が場所 l にある（フルーエント $Located(d,l)$ が `true` になる）ことを表す．式 (9) は影響公理であり，機器 d が場所 l_1 にある状態のときに異なる場所 l_2 に移動すると，場所 l_1 にある状態では無くなることを表す．

また，イベント $Borrow(s,d)$ の与える影響を式 (10) のように記述する．

$$\texttt{Initiates}(Borrow(s,d), Using(s,d), t) \tag{10}$$

式 (10) は影響式であり，スタッフ s が機器 d を借りる手続きをする（イベント $Borrow(s,d)$ が発生する）と，スタッフ s が機器 d を使用している状態になることを表す．

b） 状 態 制 約

次に，機器の位置と使用状況から，それらの情報を組み合わせたフルーエントを導く状態制約を式 (11) のように記述する．

$$\texttt{HoldsAt}(UsingAt(s,d,l),t) \Leftrightarrow \texttt{HoldsAt}(Using(s,d),t) \wedge \texttt{HoldsAt}(Located(d,l),t) \tag{11}$$

この式は，スタッフ s が機器 d を場所 l で使用している（$UsingAt(s,d,l)$ が `true` である）ことは，スタッフ s が機器 d を使用しており（$Using(s,d)$ が `true` であり）かつその機器 d が場所 l にある（$Located(d,l)$ が `true` である）ことと同義であると定義している．

3.4.3 イベントの入力と推論処理

式 (8) から式 (11) がイベント記述として登録され，かつ以下のイベントが順に発生したとする．なお，*D1* はある機器を，*N1* はある看護師（nurse）を示す．

$$Move(D1, HospitalRoom1) \tag{12}$$

$$Borrow(N1, D1) \tag{13}$$

以下，各時刻で手順 1 から 5 が適用される流れを述べる．ただし，現在の想定シナリオにおいてトリガー公理は存在しないため，手順 2 については省略する．

$t = 0$

時刻 $t = 0$ において状態制約（式 (11)）を満たすフルーエントは存在しないため，手順 1 はスキップされる．次に，手順 3 においてイベント $Move(D1, HospitalRoom1)$ がシステムに入力され，以下のイベント生起として処理される．

$$\texttt{Happens}(Move(D1, HospitalRoom1), 0) \tag{14}$$

このとき，式 (8) の影響式により，フルーエント $Located(d,l)$ の真偽値が次の時刻で `true` になることが導かれる．したがって，手順 4 で時刻を 1 進めた後，手順 5 で以下の式が得られる．

$$\texttt{HoldsAt}(Located(D1, HospitalRoom1), 1) \tag{15}$$

$t = 1$

時刻 $t = 1$ においても同様に，状態制約を満たすフルーエントは存在しないため，手順 1 はスキップされる．次に，手順 3 においてイベント $Borrow(N1, D1)$ が入力され，以下のイベント生起として処理される．

$$\texttt{Happens}(Borrow(N1, D1), 1) \tag{16}$$

このとき，式 (10) の影響式により，フルーエント $Using(s,d)$ の真偽値が次の時刻で `true` になることが導かれる．したがって，手順 4 で時刻を 1 進めた後，手順 5 で以下の式が得られる．

$$\texttt{HoldsAt}(Using(N1, D1), 2) \tag{17}$$

$t = 2$

式 (15) 及び式 (17) により状態制約（式 (11)）が満たされ，手順 1 において以下の式が導かれる．

$$\texttt{HoldsAt}(UsingAt(N1, D1, HospitalRoom1), 2) \tag{18}$$

その後，手順 3 において入力イベントが存在しないため，次のイベントが入力されるまでシステムは待機する．

処 理 結 果

処理の結果として，式 (14) から式 (18) のイベント及びフルーエントが得られる．つまり，$t = 2$ の時点で表 6 から表 13 のテーブルが RDB に記録されてる．なお，タプルの存在しないテーブルについては省略する．

表 6: Move テーブル

d	l	t
D1	*HospitalRoom1*	0

表 7: Borrow テーブル

s	d	t
N1	*D1*	1

表 8: Located テーブル

d	l	ts	te
D1	*HospitalRoom1*	0	NULL

表 9: Located_c テーブル

d	l
D1	*HospitalRoom1*

4. RDB による推論処理

RDB による推論処理について述べる．

表 10: Using テーブル

s	d	ts	te
N1	*D1*	2	NULL

表 11: Using_c テーブル

s	d
N1	*D1*

表 12: UsingAt テーブル

s	d	l	ts	te
N1	*D1*	*HospitalRoom1*	2	NULL

表 13: UsingAt_c テーブル

s	d	l
N1	*D1*	*HospitalRoom1*

4.1 公理の評価結果をもつテーブル

記述に制限を設けることで，RDB への問合せによって値を変更するフルーエントや発生するイベントの一覧，すなわちテーブル E_C，F_C へ挿入・削除するタプルの一覧を得ることができる．ここで，そのようなタプルの一覧をもつテーブル G を考える．各公理とテーブル G との関係を表 14 に示す．

表 14: 公理の形式とテーブル G の関係

公理の形式	テーブル G の内容
$\texttt{Initiates}(e,f,t)$	次の時刻に，この公理によって値が `true` になるフルーエント f の一覧．既に `true` のものも含む．
$\texttt{Terminates}(e,f,t)$	次の時刻に，この公理によって値が `false` になるフルーエント f の一覧．既に `false` のものも含む．
$\gamma \Rightarrow \texttt{Initiates}(e,f,t)$	次の時刻に，この公理によって値が `true` になるフルーエント f の一覧．既に `true` のものも含む．
$\gamma \Rightarrow \texttt{Terminates}(e,f,t)$	次の時刻に，この公理によって値が `false` になるフルーエント f の一覧．既に `false` のものも含む．
$\gamma \Rightarrow \texttt{Happens}(e,t)$	現在時刻に生起するイベント e の一覧．
$\texttt{HoldsAt}(f,t) \Leftrightarrow \gamma$	現在時刻において，値が `true` であるフルーエント f の一覧．

4.2 RDB の問合せによる処理が可能な記述

以下の条件を満たす記述については，RDB の問合せによる処理が可能である．

- 条件 γ が以下を満たす
  - 否定の式 $\neg\texttt{HoldsAt}(f,t)$ を含まない
  - 論理和 $\vee$ を含まない
  - 存在限量子 $\exists$ を含まない

なお，以下のように公理を分割することで，式 (19) の γ が論理和を含む形から，式 (20) の論理和を含まない形に変形できる．

$$\gamma_1 \vee \gamma_2 \Rightarrow \pi(e,f,t) \tag{19}$$

$$\begin{aligned}\gamma_1 &\Rightarrow \pi(e,f,t)\\ \gamma_2 &\Rightarrow \pi(e,f,t)\end{aligned} \tag{20}$$

また，$\exists x\Gamma(x)$ は $\bigvee_i \Gamma(x_i)$ で置き換えてインスタンス化することで限量子 $\exists$ を含まない形に変形できる．ただし，x_i は x のとりうる値で，$\Gamma(x)$ は x を含む式である．

4.3 SQL による記述

公理を SQL へ変換し，それを実行することで推論結果であるタプルの一覧を得る．

イベント計算の公理を用いて，影響式・影響公理は表 15 に示すように `Initiates`，`Terminates` を用いない形式で記述できる．

表 15: 影響式・影響公理の書き換え

公理の形式	`Initiates`，`Terminates` を用いない表現
$\texttt{Initiates}(e,f,t)$	$\texttt{Happens}(e,t) \Rightarrow \texttt{HoldsAt}(f,t+1)$
$\texttt{Terminates}(e,f,t)$	$\texttt{Happens}(e,t) \Rightarrow \neg\texttt{HoldsAt}(f,t+1)$
$\gamma \Rightarrow \texttt{Initiates}(e,f,t)$	$\texttt{Happens}(e,t) \wedge \gamma \Rightarrow \texttt{HoldsAt}(f,t+1)$
$\gamma \Rightarrow \texttt{Terminates}(e,f,t)$	$\texttt{Happens}(e,t) \wedge \gamma \Rightarrow \neg\texttt{HoldsAt}(f,t+1)$

すなわち，影響式・影響公理は，式 (21) または式 (22) のように書ける．影響式は，γ が常に `true` である特殊な場合と考えればよい．

$$\begin{aligned}&\texttt{Happens}(E(a_1,\ldots,a_n),t) \wedge \gamma\\ &\Rightarrow \texttt{HoldsAt}(F(b_1,\ldots,b_m),t+1)\end{aligned} \tag{21}$$

$$\begin{aligned}&\texttt{Happens}(E(a_1,\ldots,a_n),t) \wedge \gamma\\ &\Rightarrow \neg\texttt{HoldsAt}(F(b_1,\ldots,b_m),t+1)\end{aligned} \tag{22}$$

式 (21)，式 (22) のいずれも，テーブル G を得るために，入力イベントによって $a_1,\ldots,a_n$ に対する制約 $a_1 = A_1,\ldots,a_n = A_n$ が与えられたときに $b_1,\ldots,b_m$ の候補を列挙すればよい．これは，以下の SQL 問合せによって可能である．

$$\begin{aligned}&\texttt{SELECT } b_1,\ldots,b_m\\ &\texttt{FROM } E_C, F_{1C},\ldots,F_{lC}, V_1,\ldots,V_p\\ &\texttt{WHERE } \Gamma\end{aligned}$$

ただし，Γ は式中のすべての変数について，その変数間に存在する制約を表す比較式の連言である．$F_{1C},\ldots,F_{lC}$ はそれぞれ γ に出現する $\texttt{HoldsAt}(f_1,t),\ldots,\texttt{HoldsAt}(f_l,t)$ のフルーエントに対応するテーブルである．$V_1,\ldots,V_p$ はそれぞれ変数 $v_1,\ldots,v_p$ について，各変数の取りうる値（ドメイン）の一覧をもつテーブルであり，必要に応じて FROM 句に追加する．

例えば，式 (8) は以下の SQL に変換できる．

$$\begin{aligned}&\texttt{SELECT } d, l\\ &\texttt{FROM } Move_C\end{aligned}$$

トリガー公理，状態制約も同様に変換できる．例えば，式 (11) は以下の SQL に変換できる．

$$\begin{aligned}&\texttt{SELECT } s, Using_C.d, l\\ &\texttt{FROM } Using_C, Located_C\\ &\texttt{WHERE } Using_C.d = Located_C.d\end{aligned}$$

なお，実際の処理ではテーブル G を実体化ビューとして定義し，実体化ビューの効率的な更新手法 [2] により，インクリメンタルな処理を行う．

5. まとめと今後の課題

本稿では，イベント計算の技術と RDB を組み合わせた複合イベント処理をフレームワークを提案した．まず，想定するイベント計算の枠組みを定義した．次に，提案するフレームワークについて述べた．その後，RDB の問合せによる推論手法について述べた．

今後の課題としては，検出した複合イベントについての通知・問合せの手法の考案が挙げられる．システムを利用するユーザごとに監視したい複合イベントが異なるという場合が考えられるため，ユーザごとに通知するイベントを指定・管理する仕組みが必要である．また，過去の状態変化の記録に対する問合せとしてどのような要求や特徴があるのかを分析し，具体的な問合せのインタフェースの考案する必要がある．

謝辞　本研究の一部は，科研費（16H01722）による．

文　献

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. ACM PODS*, pp. 1–16, 2002.

[2] Stefano Ceri and Jennifer Widom. Deriving production rules for incremental view maintenance. In *VLDB*, pp. 577–589, 1991.

[3] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, Vol. 44, No. 3, pp. 15:1–15:62, 2012.

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

[5] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, Vol. 4, No. 1, pp. 67–95, 1986.

[6] Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In Sadri F. Kakas, A.C., editor, *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski*, Vol. 2408 of *LNCS*, pp. 452–490. Springer, 2002.

[7] Erik T. Mueller. *Commonsense reasoning*. Morgan Kaufmann, 2006.

[8] Murray Shanahan. The event calculus explained. In M. J. Woodridge and M. M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, Vol. 1600 of *LNCS*, pp. 409–430. Springer, 1999.

[9] Murray Shanahan and Mark Witkowski. Event calculus planning through satisfiability. *Journal of Logic and Computation*, Vol. 14, No. 5, pp. 731–745, 2004.