オントロジとデータベース技術を活用した複合イベント処理システム

高橋　正和†　　簗井　美咲†　　佐々木勇和††　　石川　佳治†,†††

† 名古屋大学大学院情報科学研究科
†† 名古屋大学未来社会創造機構
††† 国立情報学研究所
E-mail: † {takahashi,yanai}@db.ss.is.nagoya-u.ac.jp, ishikawa@is.nagoya-u.ac.jp
†† yuya@db.ss.is.nagoya-u.ac.jp

あらまし　近年，Web 上で生成されるデータだけでなく，実世界で生成されたデータなども含めた，様々な情報が利用可能になってきている．特に，スマートフォンなどのモバイルデバイスを利用すれば，SNS などへリアルタイムに情報を発信できるだけではなく，搭載されたセンサを使用することで，位置情報などの実世界のデータを利用することも可能である．本稿では，このようにして得られる情報をイベントと呼び，これらを処理することで，より高次な複合イベントを検出するシステムの開発を目指している．本システムでは，イベントストリームに対して，オントロジとリレーショナルデータベースの技術を活用することで，イベント検出を実現する．

キーワード　複合イベント処理, イベント検出

1. はじめに

ICT の普及により様々なデータが利用可能になってきている．Web 上では，Web ページのような静的な情報だけでなく，SNS への投稿などのリアルタイムに更新される情報を得ることが可能である．また，センサ機器の普及により実世界のデータも利用できるようになった．交通情報や気象情報などの大規模な設備から得た情報だけでなく，スマートフォンなどのモバイルデバイスに搭載された GPS や加速度センサから，各個人の位置情報や行動情報をリアルタイムに取得可能になった．

SNS への投稿や位置情報など，Web やセンサ機器で次々と生成されて無限に流れてくる時系列データを，データストリームと呼ぶ．**複合イベント処理**（complex event processing, CEP）[1] は，データストリームからより高次のイベントを検出しようとする処理である．既存の複合イベント処理における「イベント」は，数値などの単純なデータアイテムからなると想定されていた．しかし，実世界におけるアプリケーションの高度化を考えると，より高レベルの意味的なイベントを表現することが必要になる．以下では，そのようなドメインの例を挙げる．

例（位置に基づく情報サービス）

位置に基づく情報サービス（locaton-based service, LBS）では，移動するユーザや車などに対し，様々な情報サービスを提供する．近年ではさらに，ソーシャルネットワークサービスと位置情報を連携した，位置に基づくソーシャルネットワーク（locaton-based social network, LBSN）[2] も発展しているが，これも LBS の一種といえる．

LBS や LBSN では様々なイベントを扱う必要がある．例えば，モバイルユーザ同士の出会いを支援する LBSN においては，基本的なイベントとして「ユーザの存在（ユーザがある時刻にある場所にいた）」というものが考えられる．このイベントは，既存の SNS にあるチェックイン機能に類似したイベントで，位置情報（緯度・経度）だけでなく，ユーザや訪れた場所に関する情報などの様々な情報が含まれており，これらを表現する必要がある．また，複数人の「ユーザの存在」イベントを組み合わせることで,「位置が近い二人」という高レベルのイベントを検出することができる．さらに,「二人のユーザが友人である」という背景知識を利用すれば，より高レベルのイベント「友人同士が近くにいる」が検出可能である．しかし，友人関係や近さの関係は，応用分野や実際に構築するアプリケーションに大きく依存しており，必ずしも固定することはできない．そのため，応用に応じた多様なイベントを扱えることが重要となる．

以上の例に示したような応用領域を考えた場合，高度なイベント処理を行うには以下のような機能が必要となる．

- アプリケーションに応じてイベントを定義
- イベント定義に基づき，複合イベントを検出・処理

高レベルの意味的なイベントを表現するために，**Semantic Web** 技術 [3] を活用した**オントロジ**を利用する．オントロジを利用することで，複雑な知識体系の表現と意味的な処理が可能となる．また，公開されている既存のオントロジを再利用することもできる．オントロジは，データを主語・述語・目的語のトリプルで表現する枠組みである **RDF**（Resource Description Framework）で表現する．

本研究では，オントロジを活用することでより高レベルの意味的なイベント検出を行うことができるシステムを開発する．入力イベントは RDF 形式で表現され，RDF 形式に基づくデータストリームが与えられると想定する．本システムでは，既存の RDF ストリーム処理エンジンとリレーショナルデータベースの問合せ技術を活用することで，イベント検出処理を実現する．

2. 関 連 研 究

2.1 RDF ストリームに対する問合せ

本研究における入力イベントストリームは RDF 形式のデータストリームを想定しており，処理に必要なイベントを検出するためには RDF ストリームに対する連続的な問合せを行う必要がある．静的な RDF データに対しては，SPARQL [4] という，SQL に類似の言語で問合せを行うことができる．この SPARQL を RDF データストリーム上の連続的問合せのために拡張したものが C-SPARQL [5] である．C-SPARQL では，RDF ストリームに対して時間窓（window）を指定して問合せを実行する機能や問合せ結果を RDF ストリームに整形する機能が追加されている．

C-SPARQL エンジンは公開されており，実際に利用することができる (注1)．しかし，このエンジンでは，単純なパターンマッチ，フィルタリング，結合は実行可能だが，複雑な集約・推論処理は実行できない [6], [7] ．また，ストリーム処理エンジンは，それぞれが異なるセマンティクスに従って設計されているため，時間窓の動作や出力方式などが異なる [8]．そのため，C-SPARQL エンジンで実現可能な処理と実際の動作を十分考慮して利用する必要がある．

2.2 セマンティックな地理空間情報を扱うシステム

本研究の類似研究として， RDF 形式の地理データストリームに対する問合せを行うオープンソースツールキット Semantic GeoStreaming Toolkit（SGST）[9] がある．

このツールキットが提供する主な機能には，セマンティックな地理データストリームの管理，様々なソースからのリアルタイムな地理データストリームの取得などがある．利用例としては，リアルタイムの地震情報（位置とマグニチュード）を取得してデータベースに蓄積し，それらに対して時空間問合せ（例：指定した時期，エリアで発生した地震を検出）を実行するというものが挙げられている．

本研究との共通点は，RDF 形式の地理データストリームをリレーショナルデータベースに格納してから処理を行っている点である．しかし，連続的問合せや検出結果を組み合わせた高レベルなイベント検出を行っていない点が本研究とは異なる．

3. イベントの定義

本研究で扱うイベントの定義について述べる．イベントの表現については，オントロジ記述の枠組みとして RDF を使用する一方，一般的なオントロジの記法からは逸脱するがルールベース的な表現も用いてイベント記述を行う [10].

3.1 プリミティブイベント

それ以上分解できない最も低レベルのイベントを **プリミティブイベント**（primitive event, PE）と呼ぶ．プリミティブイベントは，RDF 形式で表現され，入力イベントストリームを構成している．一例として,「時刻 T において人物 P が位置 L に存在した」というプリミティブイベントを考え，これを

(注1)：http://streamreasoning.org/download

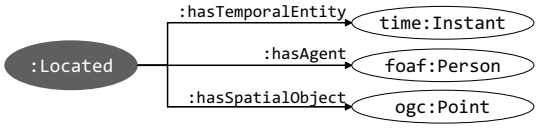

図 1　プリミティブイベント *Located* の定義

$$Located(P, L, T)$$

と表現する．*Located* を RDF で表現すると図 1 のようになる．

また，プリミティブイベントの引数を一部指定することで定義されるイベントもプリミティブイベントとして扱う．例えば，'User' に関する *Located* を 表すイベントは，

$$MyLocated(L, T) := Located('User', L, T)$$

と表現することができ，*MyLocated* もプリミティブイベントである．

3.2 複合イベント

プリミティブイベントと背景知識などのデータを組み合わせることで検出されるイベントを**複合イベント**（complex event, CE）と呼ぶ．複合イベントは，ルールを用いて定義する．例えば,「2 人の人物がある時点で近くにいる」という複合イベント *ClosePeople* は，以下のようなルールを用いて定義する．

$$\begin{aligned} &ClosePeople(P_1, P_2, L_1, L_2, T_1, T_2) \\ &\quad := Located(P_1, L_1, T_1) \wedge Located(P_2, L_2, T_2) \\ &\qquad \wedge NearLoc(L_1, L_2) \wedge NearTime(T_1, T_2) \end{aligned}$$

ここで，$NearLoc(L_1, L_2)$ および $NearTime(T_1, T_2)$ はプリミティブイベントではなく，それぞれ位置と時刻が近い場合に真になる述語であり，具体的な近さの基準はアプリケーションごとにオントロジを用いて定義される．

複合イベントからさらに高次の複合イベントを定義することもできる．

前述の *ClosePeople* をもとに,「**友人同士**である 2 人の人物がある時点で近くにいる」複合イベント *CloseFriends* は，以下の様なルールで定義する．

$$\begin{aligned} &CloseFriends(P_1, P_2, L_1, L_2, T_1, T_2) \\ &\quad := ClosePeople(P_1, P_2, L_1, L_2, T_1, T_2) \\ &\qquad \wedge Friends(P_1, P_2) \end{aligned}$$

ここで，$Friends(P_1, P_2)$ は 2 人が友人である場合に真になる述語であり，友人関係に関する情報が格納されているデータベースを利用した処理を行うことを想定している．

3.3 イベントの有効期間と更新間隔

イベントには有効期間と更新間隔が指定されている．

イベントの有効期間は検出したイベントを保持する期間のことで，以下の 3 種類のポリシーが考えられる．

A. 明示的に削除されるまで，半永久的に保持
B. 時間窓の設定に応じて，一定期間だけ保持
C. 検出し，報告したら即座に破棄

また，更新間隔はイベント検出処理を行う間隔のことで，以下の 3 種類から指定する．

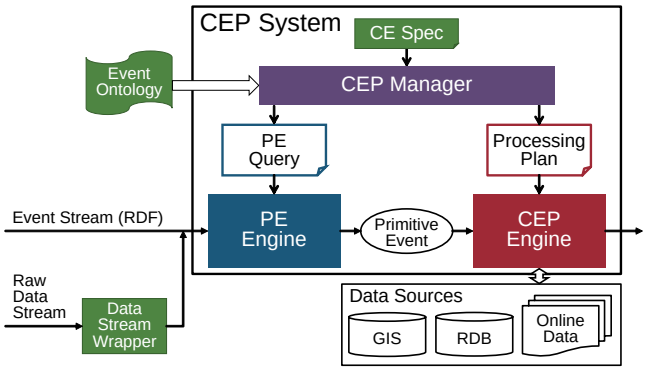

図 2　システムのアーキテクチャ

a.　報告の要求があったときのみ更新
b.　定期的に更新
c.　プリミティブイベントが検出される毎に更新

例えば，あるイベントの有効期間を 30 分間保持，更新間隔を 10 分間隔としたとき，イベント検出処理は 10 分間隔で行われ，検出されたイベントは 30 分間保持される．

4.　システムアーキテクチャ

本研究で開発するシステムのアーキテクチャを図 2 に示す．

複合イベント検出処理の中心となるのが，**複合イベント処理システム**（complex event processing system）である．複合イベント処理システムを略して **CEP システム**（CEP system）と呼ぶ．CEP システムに対する入力は，RDF 形式のイベントストリームを想定している．そのため，センサデータなどの RDF 形式以外のデータについては，**データストリームラッパ**（data stream wrapper）で RDF ストリームに変換してシステムに入力する．例えば GPS から取得した位置情報のデータストリームを考えると，流れてくる位置情報を RDF 形式に変換するのに加え，時刻や GPS が搭載されている機器に関する情報など，アプリケーションに応じて必要な情報を付加して，イベントストリームを生成する．

また，CEP システムは必要に応じてさまざまな情報を外部のデータソースから取得する．取得するデータには，地理情報をもつ GIS やデータベースなどの開発者が定義したローカルデータに加え，オンライン上に存在するデータを取得することも想定する．

4.1　複合イベント指定

複合イベント指定（complex event specification, CE Spec）では，CEP システムでモニタリングすべき複合イベントが指定される．複合イベント指定では，イベントオントロジで定義されているプリミティブイベントと複合イベントを使用したルールを作成することでイベントを指定する．また，複合イベント指定では検出したイベントの有効期間と更新間隔を 3.3 のポリシーに従って指定する．

4.2　CEP マネジャ

複合イベント指定は，ユーザから直接的に，もしくはアプリケーションから API を通じて，**CEP マネジャ**（CEP manager）に与えられる．CEP マネジャは，複合イベント指定をもとにどのようにイベント検出を行うかをプランニングし，**プリミティブイベント問合せ**（primitive event query）および **CEP 処理プラン**（CEP processing plan）を構築する．プリミティブイベント問合せを略して **PE 問合せ**（PE query）と呼ぶ．

プランニング処理では，複合イベント指定に含まれている複合イベントを**イベントオントロジ**（event ontology）を用いて，プリミティブイベントに分解する．イベントオントロジには，3. で示したようなイベントが定義されている．また，構築するアプリケーションに応じて，イベントオントロジを追加することもある．

プリミティブイベントを入力ストリームから抽出するための連続的問合せが PE 問合せであり，C-SPARQL で記述される．CEP 処理プランには，プリミティブイベントの定義と条件判定を行う述語（例：*NearLoc*）の処理内容，複合イベントを検出するための問合せが含まれており，SQL で記述される．

4.3　PE エンジン

CEP マネジャによって生成された PE 問合せは**プリミティブイベントエンジン**（primitive event engine）に渡される．プリミティブイベントエンジンを略して **PE エンジン**（PE engine）と呼ぶ．PE エンジンの役割は，入力して得られる RDF 形式のイベントストリームに対して連続的な問合せを実行し，プリミティブイベントを検出することである．このとき，データストリームをシーケンシャルにスキャンすることで処理可能なもののみを実行し，外部のデータソースにある情報を利用した処理は行わない．検出されたプリミティブイベントは逐次 CEP エンジンに送られる．また，PE エンジンは C-SPARQL エンジンを利用して構築する．

4.4　CEP エンジン

CEP エンジン（CEP engine）は，PE エンジンからプリミティブイベントを受け取り，CEP 処理プランに従って複合イベントを検出する．CEP エンジンはリレーショナルデータベースを利用して構築する．本システムでは，リレーショナルデータベースシステムとして PostgreSQL (注2) を使用し，地理空間処理を行うために PostGIS (注3) を使用して拡張している．

リレーショナルデータベースの枠組みを用いる利点には，問合せ処理の効率化だけでなく，見通しが立てやすいことがある．特に，複合イベントを**実体化ビュー**（materialized view）として表現することは有効であると考えられる．複合イベントのルールは，**SPJ 問合せ**（select-project-join query）に対応しており，インクリメンタルな実体化ビューの管理が容易に実現できる．また，地理空間処理を行う場合，RDF 形式で行うよりも高速に処理することが可能である [11]．

CEP エンジンでは，プリミティブイベントをテーブルに格納し，それらから複合イベントを表現する実体化ビューを生成する．実体化ビューをイベントの有効期間や更新間隔に応じてリフレッシュすることで，連続的問合せ処理を実現する．

(注2)：http://www.postgresql.org/
(注3)：http://postgis.net/

Input: 複合イベント指定で定義された全複合イベント (CE)

1: **for all** $ce \in CE$ **do**
2: 　複合イベント登録 (ce)
3: **end for**
4: **procedure** 複合イベント登録 (ce)
5: 　$E \leftarrow ce$ のルールにある全要素
6: 　**for all** $e \in E$ **do**
7: 　　**if** e が未登録の複合イベント **then**
8: 　　　複合イベント登録 (e)
9: 　　**else if** e が未登録のプリミティブイベント **then**
10: 　　　PE エンジンに e を抽出する PE 問合せを登録
11: 　　**else if** e が未登録の述語 **then**
12: 　　　CEP エンジンに e を実行するユーザ定義関数を登録
13: 　　**end if**
14: 　**end for**
15: 　**if** ce の有効期間が即座に破棄 $\vee$ 更新間隔が要求時のみ **then**
16: 　　CE を格納するビューを作成する問合せを CEP エンジンに登録
17: 　**else**
18: 　　CE を格納する実体化ビューを作成する問合せを CEP エンジンに登録
19: 　**end if**
20: **end procedure**

図 3　CEP マネジャにおける処理のアルゴリズム

5. 問合せ処理

本システムにおける問合せ処理の概要について述べる．ここでは，3. で定義したプリミティブイベント *Located* から複合イベント *CloseFriends* を検出する処理を例に，問合せ処理の流れを述べる．

5.1 CEP マネジャにおける処理

CEP マネジャにおける処理のアルゴリズムを図 3 に示す．

はじめに，CEP マネジャはモニタリング複合イベント指定を入力として受け取り，全ての複合イベントを登録する（1 - 3 行目）．ここでは，モニタリング対象の複合イベントを *CloseFriends*，有効期間と出力間隔をともに 10 分として説明する．複合イベント指定で指定された複合イベントを定義に従って分析し，検出に必要なプリミティブイベントや述語，他の複合イベントを特定する（5 行目）．3. で示した *CloseFriends* のルールを参照し，*ClosePeople* と *Friends* を得る．*ClosePeople* は複合イベントであるので，さらにその定義を登録する（8 行目）．*Friends* は述語であるので，イベントオントロジから具体的な処理内容を取得し，ユーザ定義関数として CEP エンジンに登録する（12 行目）．*ClosePeople* ルールからは，$Located, NearLoc, NearTime$ が得られる（5 行目）．*Located* はプリミティブイベントであるので，イベントオントロジから定義を取得し，PE エンジンに *Located* を抽出するための PE 問合せを登録する（10 行目）．$NearLoc, NearTime$ は *Friends* と同様に述語であるため，ユーザ定義関数として CEP エンジンに登録する（12 行目）．処理に必要な PE と述語を全て特定して PE エンジンと CEP エンジンに問合せを登録した後，複合イベントを格納する実体化ビューを作成するための問合せを登録する（16 行目）．ただし， 複合イベントの有効期間が， 4.1 の検出し，報告したら即座に破棄の場合は結果を保持する必要が無いため，実体化ビューの代わりにビューを作成する問合せを登録する．また，更新間隔が報告の要求があったときにみ更新の場合についても，要求されたときに処理をすればよいので，ビューを作成して処理を行う（18 行目）．登録の順序はルールをたどったときの逆となり，*ClosePeople*， *CloseFriends* の順で CEP エンジンに登録する．問合せ内容の詳細は後述する．なお，4.2 で述べた CEP 処理プランには，具体的にはユーザ定義関数登録と実体化ビュー登録を行うための問合せが含まれている．

```
Located1  rdf:type                lbsn:located ;
          lbsn:hasTemporalEntity  Instant1 ;
          lbsn:hasAgent           User1 ;
          lbsn:hasSpatialObject   Point1 .
Instant1  rdf:type                time:Instant ;
          time:inXSDDateTime
              “2015-01-30T11:55:00”^^xsd:dateTime.
Point1  rdf:type        geo:Point ;
        geo:asWKT
              "POINT(136.966 35.155)"^^sf:wktLiteral.
```

図 4　プリミティブイベント *Located* の RDF 記述例

複合イベント指定で指定された複合イベントの有効期間と更新間隔は，PE 問合せの時間窓指定と実体化ビューのリフレッシュ操作の制御によって実現する．*CloseFriends* の場合，PE 問合せで 10 分の幅で 10 分毎にスライドする時間窓を指定し，PE エンジンが検出されるごとに全ての実体化ビューをリフレッシュすることで複合イベント指定で指定された条件を実現する．問合せ処理におけるイベントの有効期間と更新間隔の実現方法は，5.4 で詳しく述べる．

5.2 PE エンジンにおける処理

PE エンジンにおける入力は，プリミティブイベント *Located* を含むイベントストリームである．入力イベントは RDF の N-Triples 形式に基づくデータあり，*Located* の具体的な記述は図 4 のようになる．`Located1` には，時間（`Instant1`），ユーザ名（`User1`），位置情報（`Point1`）が含まれている．`Instant1` と `Point1` の具体的な情報は，時間や地理空間に関するオントロジに従って定義する [12], [13].

PE エンジンでは，入力 RDF ストリームに対し連続的な問合せを実行し，プリミティブイベント *Located* を抽出する．この問合せを C-SPARQL の構文を用いて表したのが図 5 である．

`FROM STREAM` 句では RDF ストリームを取得し，`[RANGE 10m STEP 10m]` で 10 分の幅で 10 分毎にスライドする時間窓を設定している．時間窓内のイベントに対し，`WHERE` 句で図 4 のような *Located* イベントの定義を記述することで，グラフマッチングにより *Located* イベントを抽出している．そして `SELECT` 句で， *Located* イベントの述語 $Located(P, L, T)$ の引数である P, L, T に相当する `?person, ?loc, ?time` を選択している．そして，問合せ結果は CEP エンジンの *Located*

```
SELECT ?person ?location ?time
FROM STREAM <http://lbsn.org/stream/>
                                [RANGE 10m STEP 10m]
WHERE {
    ?located    lbsn:hasTemporalEntity  ?instant;
                lbsn:hasAgent           ?person ;
                lbsn:hasSpatialObject   ?point .
    ?instant    rdf:type                time:Instant;
                time:inXSDDateTime      ?time .
    ?point      rdf:type                geo:Point ;
                geo:asWKT               ?location .
}
```

図 5　PE 問合せの例

表 1　*Located* テーブル

P	L	T
User1	POINT(136.966 35.155)	2015-01-30 11:55:00
User1	POINT(136.963 35.155)	2015-01-30 12:00:00
⋮	⋮	⋮

```
CREATE MATERIALIZED VIEW closepeople AS
SELECT DISTINCT x.P AS P1, y.P AS P2, x.L AS L1,
                y.L AS L2, x.T AS T1, y.T AS T2
FROM    Located x, Located y
WHERE   NearLoc(x.L, y.L) AND NearTime(x.T, y.T)
```

図 6　*ClosePeople* ビューを作成する問合せ

```
CREATE MATERIALIZED VIEW closefriends AS
SELECT DISTINCT P1, P2, L1, L2, T1, T2
FROM    closepeople
WHERE   Friends(P1, P2)
```

図 7　*CloseFriends* ビューを作成する問合せ

テーブルに格納される（表 1）.

5.3　CEP エンジンにおける処理

CEP エンジンでは，プリミティブイベントから複合イベント指定で指定された複合イベントを検出する．ここでは，プリミティブイベント *Located* から複合イベント *ClosePeople* を検出し，さらに処理を行うことで指定された複合イベント *CloseFriends* を検出する．

5.3.1　PE からの CE の検出

テーブルに挿入されたプリミティブイベントから複合イベントを検出し，実体化ビューを生成する．プリミティブイベント *Located* から複合イベント *ClosePeople* の実体化ビューを生成する問合せを図 6 に示す.

`NearLoc`，`NearTime` はユーザ定義関数で，あらかじめ CEP マネジャによって CEP エンジンに登録されているものとする.

5.3.2　CE からの CE の検出

生成した実体化ビューと他のテーブルとの結合処理を行うことで，より高次な複合イベントを検出する．複合イベント *ClosePeople* の実体化ビューと友人関係の情報を結合して，複合イベント *CloseFriends* を生成する問合せを図 7 に示す.

`Friends` は友人か否か判定するユーザ定義関数で，友人関係の情報をもつテーブルを参照している．よって，この問合せは `closepeople` イベントが格納されている実体化ビューと `Friends` が参照しているテーブルの結合処理を行っている.

表 2　イベントの有効期間を実現するための処理

有効期間	PE	CE
A（半永久的）	削除しない	削除しない
B（一定期間）	期間外イベントを削除	問合せに条件を追加
C（即座に破棄）	全イベント削除	通常のビューを作成

表 3　イベントの更新間隔を実現するための処理

更新間隔	PE（時間窓の間隔）	CE
a（報告要求時）	事前に定義 (注4)	通常のビューを作成
b（定期的）	更新間隔と同じ	更新間隔でリフレッシュ
c（PE 検出毎）	事前に定義 (注5)	PE 検出時にリフレッシュ

5.4　イベントの有効期間と更新間隔の実現方法

5.4.1　有効期間の実現方法

イベントの有効期間を実現する処理では，テーブルや実体化ビューから有効期間外の不要になったイベントを削除を行う．複合イベントについては，格納されている実体化ビューを生成する問合せに，有効期間内のイベントを取得する条件を追加することで，更新される毎に不要になったイベントが削除されるようにする．また，前述のとおり，有効期間が即座に破棄の場合には，実体化ビューではなく，通常のビューを生成する．通常のビューでは問合せ毎に処理が行われて結果が保存されないため，即座に破棄するというポリシーを満たすことができることに加え，実体化ビューのように複合イベントを格納する領域を必要しない.

プリミティブイベントについては，格納されているテーブルが更新，すなわち PE エンジンから PE イベントが抽出されるごとに DELETE 文を実行することで不要なイベントを削除する.

3.3 で示した有効期間の 3 つのポリシーそれぞれにおける，有効期間を実現するための処理を表 2 に示す.

5.4.2　更新間隔の実現方法

プリミティブイベントの更新間隔については，PE 問合せの時間窓がスライドする間隔（`STEP`）を指定することで実現する．時間窓のスライド間隔は PE エンジンの出力間隔となり，その結果 CEP エンジンの PE イベントを格納したテーブルが指定された間隔で更新される.

複合イベントの更新間隔については，実体化ビューのリフレッシュ間隔を指定することで実現する．ただし，更新間隔が要求時のみの場合，通常のビューを生成するため，リフレッシュの実行は行わない.

3.3 で示した有効期間の 3 つのポリシーそれぞれにおける，更新間隔を実現するための処理を表 3 に示す.

(注4)：要求の頻度に合わせて適切な値を定義する.
(注5)：C-SPARQL の時間窓における最小のスライド間隔は 1ms.
ただし，スライド間隔内で処理が終了するような値を設定する必要がある.

6. ユースケースに基づく検証

6.1 検証の概要

同行者を考慮した行動監視というシナリオに基づき，システムを検証する．このシナリオは，監視対象者の位置情報を取得し，同行者に応じて変化する許可された区域の外に出た場合に警告を出すというものである．監視対象者は携帯端末を所持している子どもで，その保護者や友人も同様の端末を所持しており，それぞれの位置情報を取得することを想定している．

入力ストリームは，監視対象者とその友人，保護者の位置情報とする．ここから同行者の有無を検出し，その結果に応じて以下のような条件で許可区域にいるかどうかを判定し，許可された区域の外にいる場合には警告イベントが生成される．

- 一人でいるとき：自身の許可区域のみ
- 友人と同行しているとき：自身と友人の許可区域
- 保護者と同行しているとき：制限なし

データソースは，友人関係が格納されている $FriendsList$ と監視対象者や友人が許可された区域が格納されている $MyArea$ を使用する．

以上のシナリオにおいて，次の 2 点について検証を行う．

（1） 警告イベントが記述できる

（2） 現実的な時間で処理できる

6.2 検出するイベント

本システムで検出するイベントの定義を示す．

6.2.1 プリミティブイベント

入力ストリームは 3.1 で定義した $Located(P, L, T)$ を利用し，監視対象者（User）と保護者（Guardian）の $Located$ をそれぞれ，

$$MyLocated(L, T) := Located('User', L, T)$$
$$GuardianLocated(L, T) := Located('Guardian', L, T)$$

と定義する．

6.2.2 複合イベント

友人の $Located$ を抽出したイベントは，

$$friendsLocated(P, L, T) := Located(P, L, T) \wedge Friends('User', P)$$

と定義する．このイベントの内容はプリミティブイベントと類似しているが，述語 $Friends$ で友人関係のデータを参照しているため，複合イベントとして定義する．

監視対象者が保護者や友人と同行していることを表すイベントは，それぞれ以下のように定義する．

$$withGuardian(L_m, T_m) := MyLocated(L_m, T_m) \wedge GuardianLocated(L_g, T_g) \wedge NearLoc(L_m, L_g)$$

$$withFriends(L_m, T_m, P_f) := MyLocated(L_m, T_m) \wedge friendsLocated(P_f, L_f, T_f) \wedge NearLoc(L_m, L_f)$$

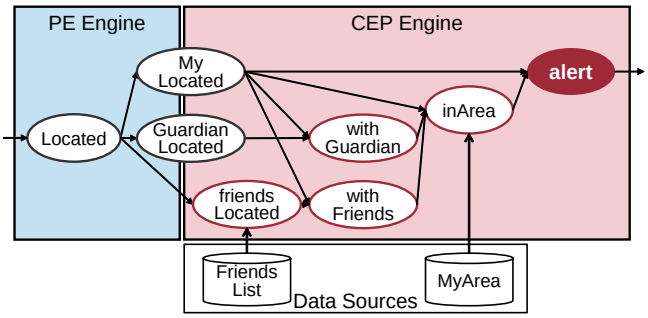

図 8 イベント検出処理の流れ

```
CREATE MATERIALIZED VIEW friendslocated AS
SELECT DISTINCT P, L, T
FROM    located
WHERE   Friends('User', P)
```

図 9 $friendsLocated$ ビューを作成する問合せ

```
CREATE MATERIALIZED VIEW withguardian AS
SELECT DISTINCT m.L, m.T
FROM    mylocated m, guardianlocated g
WHERE   NearLoc(m.L, g.L)
```

図 10 $withGuardian$ ビューを作成する問合せ

次に，監視対象者が許可された区域内に存在することを表すイベントを以下のように定義する．

$$inArea(L, T) := MyLocated(L, T) \wedge Covers('User', L)) \vee withFriends(L, T, P) \wedge Covers(P, L) \vee withGuardian(L, T)$$

$Covers(P, L)$ は，P が許可された区域に L が含まれる場合に真になる述語である．1 行目で監視対象者が許可された区域に存在するかどうか，2 行目では友人と同行しているために広がった区域に存在するかどうかを判定している．そして，3 行目は保護者といる場合で，どの区域に存在してもよいため，常に区域内と判定される．

最後に，監視対象者が許可された区域の外に出たことを警告するイベントを，

$$alert(L, T) := MyLocated(L, T) \wedge \neg InArea(L, T)$$

と定義する．ここでは，$MyLocated$ のうち，$InArea(L, T)$ に含まれていないものを検出することで，区域外に出たことを判定する．本シナリオではこの $alert$ イベントを複合イベント指定で定義し，有効期間と更新間隔はともに 10 分とする．

6.3 問合せ処理

5. で述べたように，はじめに CEP マネジャが複合イベント指定を分析し，検出すべきイベントを特定する．特定したイベントとそれらを検出する処理の流れを図 8 に示す．青枠の楕円はプリミティブイベント，赤枠の楕円は複合イベント，赤色の楕円は複合イベント指定した複合イベントを表している．

次に，CEP マネジャは各イベントを検出するための問合せを生成し，PE エンジンと CEP エンジンに登録する．プリミティブイベントについては，5.2 と同様または類似の処理を行う．

```
CREATE MATERIALIZED VIEW withfriends AS
SELECT DISTINCT m.L, m.T, f.P AS F
FROM    mylocated m, friendslocated f
WHERE   NearLoc(m.L, g.L)
```

図 11　*withFriends* ビューを作成する問合せ

```
CREATE MATERIALIZED VIEW inarea AS
SELECT DISTINCT L, T
FROM    mylocated
WHERE   Covers('User', L)
UNION
SELECT DISTINCT L, T
FROM    withfrineds
WHERE   Covers(F, L)
UNION
SELECT DISTINCT L, T
FROM    withguardian
```

図 12　*inArea* ビューを作成する問合せ

```
CREATE MATERIALIZED VIEW alert AS
SELECT DISTINCT L, T
FROM    mylocated
WHERE NOT EXISTS (
  SELECT *
  FROM inarea
)
```

図 13　*alert* ビューを作成する問合せ

複合イベント *friendsLocated* に関する実体化ビューを生成する問合せを図 9 に示す．`Friends` は 5.3.2 で示したものと同様のユーザ定義関数である．

withGuardian，*withFriends* に関する問合せをそれぞれ図 10，11 に示す．これら問合せは，5.3.1 の *ClosePeople* に関する問合せ（図 6）に類似しているが，ユーザ定義関数 `NearTime` を使用していない．これは，イベントの有効期間を使用して有効なイベント同士は時間的に近いと判断しているためである．

inArea に関する問合せを図 12 に示す．定義における ∨（OR）は，`UNION` を用いて実現している．

alert に関する問合せを図 13 に示す．定義における ¬（NOT）は，`NOT EXISTS` を用いて実現している．

以上の問合せをエンジンに登録した後，入力ストリームを受け取って処理を開始する．複合イベント指定で定義された *alert* の有効期間と更新間隔はともに 10 分であるので，PE エンジンは 10 分毎にプリミティブイベントを抽出し，CEP エンジンはプリミティブイベントを受け取る毎に図 8 で示した実体化ビューを左側に示したものから順にリフレッシュすることでイベント検出処理を行う．

6.4　性能評価

本システムの性能を評価するために，実行時間の測定を行う．実験で用いた計算機のスペックを表 4 に示す．

入力イベントは，監視対象ユーザと保護者，3 人の友人，及び他人の *Located* イベントで構成されており，ユーザのイベント近くには，他の 5 人のいずれかのイベントが存在しているものとする．評価 1 における入力イベントの構成を表 5 に示す．

表 4　実行マシンのスペック

OS	Windows7 Professional 64bit
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz
メモリ	8GB
HDD	SEAGATE ST1000DM003 7200rpm
RDB	PostgreSQL 9.3.5
GIS	PostGIS 2.1.3

表 5　入力イベントの構成

イベント総数	ユーザ	保護者	友人	他人
2000	1000	200	600	200
4000	2000	400	1200	400
6000	3000	600	1800	600
8000	4000	800	2400	800
10000	5000	1000	3000	1000

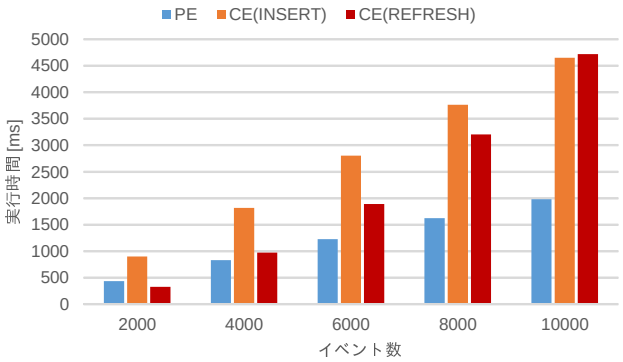

図 14　イベント数と実行時間

この条件におけるイベント総数と実行時間の関係を図 14 に示す．PE は PE エンジンにおけるプリミティブイベント検出の実行時間，CE(INSERT) は PE エンジンの結果を CEP エンジンのデータベースに挿入する処理の実行時間，CE(REFRESH) は CEP エンジンにおけるリフレッシュ，すなわち複合イベント検出処理の実行時間を表す．

PE と CE(INSERT) は，イベント数が増加に伴って実行時間が線形に増加している．これらの処理はイベントの内容にかかわらず，プリミティブイベントを抽出してデータベースに挿入するという処理を行うため，このような結果になったと考えられる．一方，CE(REFRESH) は，イベント数の二乗に比例して実行時間が増加している．

6.5　検証結果

警告イベントの記述については，6.2 で複合イベントの組み合わせによるルールで記述し，6.3 で複合イベントを表すルールに対応した実体化ビューを生成することによって，複合イベント処理を実現した．

また，6.4 の結果より，本システムにおける複合イベント処理は，イベント数が 10000 の場合でも現実的な時間で処理可能であった．これは，リレーショナルデータベースを利用することにより，データソースとの結合や地理空間処理における，効率的な問合せによって実現されたと考えられる．

7. 議　　論

7.1 検証システムにおける今後の見通し

7.1.1 イベントの優先度

行動監視システムにおいては，近くの危険情報を本人や保護者に通知する機能が要求されると考えられる．これは，危険情報の内容，発生時刻，位置をプリミティブイベントとして定義し，それがユーザの近くであるかを判定することで実現できる．

しかし，本システムではイベント検出を下位から順に検出していくため，優先すべき危険情報の通知が遅れてしまう可能性がある．そのため，複合イベント指定において複合イベントの優先度を指定できるようにすべきであると考える．

7.1.2 入力が存在しない場合の処理

監視対象者から一定時間入力がない場合，何らかの異常事態が発生したと考えて警告を行う必要がある．この機能はイベントを格納するテーブルにデータが存在を確認することで実現できる．しかし，更新間隔を実現している PE エンジン（C-SPARQL）は入力が存在しない場合に結果を出力しないため，テーブルを更新することができない．

この問題は，CEP エンジンにおける更新間隔を CEP マネジャで実現することで解決できる．CEP マネジャがイベントの有効期間と更新間隔を管理することで，PE エンジンの処理に影響を受けずに処理が可能となる．

7.2 汎用的なシステムにおける課題

7.2.1 有効期間と更新間隔のポリシー

イベントの有効期間と更新間隔のポリシーは，検出される全てのイベントに適用されると想定しているが，汎用的なシステムでは各イベントに異なるポリシーを設定する必要が存在する．各イベントでポリシーを設定可能とするために，CEP エンジンのイベントを CEP マネジャによって管理する機能を追加する．

一部のイベントのみ設定されている場合，他のイベントのポリシーを自動的に設定しなければならない．ポリシーを自動的に設定する場合，最も高レベルのイベントについてはポリシーが設定されているものとする．このとき，下位のイベントには上位のイベントと同じポリシーが設定される．上位のイベントが複数存在するとき，有効期間はより長い方，更新間隔はより短い方のポリシーを設定する．

7.2.2 状態を表すイベント

本稿で想定しているイベントは，一度生成されたら変更されない．しかし，実世界のより複雑なドメインを想定すると，ユーザや施設の状態を保持し，他のイベントによって状態が変化するイベントが必要となる [14]．例えば，状態を表すイベント「所持金」は,「購入」イベントによって変化し，所持金が少なくなったら通知するというシナリオを実現できる．

状態を表すイベントを処理するためには，状態を保持するためのテーブル作成，状態を変化させるイベントに対する処理，状態の変化からイベントを検出する処理の機能が必要である．また，状態を表すイベントは 3. で述べたような方法でで表現することが難しい．したがって，状態を表すイベントを処理するためには，システムの設計全体にわたる拡張が必要となる．

8. ま と め

本稿では，オントロジとリレーショナルデータベースを利用した意味的なイベント処理システムの設計と開発を行った．開発したシステムでは，C-SPARQL エンジンを利用して RDF 形式のデータストリームからイベントを抽出し，それらをリレーショナルデータベースを利用して他のイベントやデータと組み合わせることで，複合イベント処理を実現している．

本システムで使用している C-SPARQL や PostgreSQL, PostGIS の開発は現在も続けられているため，仕様の変更や機能の追加が行われる可能性がある．今後はこれらの動向を注視しつつ，7. で述べたシステムの拡張を行う予定である．

謝　　辞

本研究は科研費（25280039, 26540043）による．

文　　献

[1] G. Cugola and A. Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, Vol. 44, No. 3, 2012.

[2] C.-Y. Chow, J. Bao, and M. F. Mokbel. Towards Location-based Social Networking Services. In *International Workshop on Location Based Social Networks (LBSN '10)*, pp. 31–38, 2010.

[3] 荒木雅弘. フリーソフトで学ぶセマンティック Web とインタラクション. 森北出版, 2010.

[4] SPARQL 1.1 overview. http://www.w3.org/TR/sparql11-overview/, March 2013.

[5] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Querying RDF Streams with C-SPARQL. *SIGMOD Rec.*, Vol. 39, No. 1, pp. 20–26, 2010.

[6] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, and M. Fink. Linked Stream Data Processing Engines: Facts and Figures. In *International Semantic Web Conference (ISWC '12)*, pp. 300–312. Springer, 2012.

[7] Y. Zhang, P. M. Duc, O. Corcho, and J.-P. Calbimonte. SR-Bench: A Streaming RDF/SPARQL Benchmark. In *International Semantic Web Conference (ISWC '12)*, pp. 641–657. Springer, 2012.

[8] D.Dell' Aglio, J.-P. Calbimonte, M. Balduini, O. Corcho, E. Della Valle. On Correctness in RDF stream processor benchmarking. In *International Symposium on Wearable Computers (ISWC '13)*, pp. 326–342. Springer, 2013.

[9] J. Lee, Y. Liu, and L. Yu. SGST: An Open Source Semantic Geostreaming Toolkit. In *ACM SIGSPATIAL International Workshop on GeoStreaming (IWGS '11)*, pp. 17–20. ACM, 2011.

[10] 簗井美咲, 高橋正和, 佐々木勇和, 石川佳治. LBSN オントロジの構築. 第 13 回情報科学技術フォーラム（FIT 2014）, 2014.

[11] K. Patroumpas, G. Giannopoulos, and S. Athanasiou. Towards GeoSpatial Semantic Data Management: Strengths, Weaknesses, and Challenges Ahead. *ACM SIGSPATIAL*, 2014.

[12] Time Ontology in OWL. http://www.w3.org/TR/owl-time/.

[13] Wikipedia: Simple Features. http://en.wikipedia.org/wiki/Simple_Features.

[14] G. Guizzardi, G. Wagner, R. de Almeida Falbo, R. SS Guizzardi, and J. P. A Almeida. Towards Ontological Foundations for the Conceptual Modeling of Events. In *Conceptual Modeling*, pp. 327–341. Springer, 2013.