DEIM Forum 2011 B8-3

実世界トレーサビリティの為のクエリ処理システムの提案

鈴木　克弥† 　沼尾　雅之†

† 電気通信大学 電気通信学研究科 情報工学専攻　〒 182–8585 東京都調布市調布ヶ丘 1–5–1
E-mail: †{skat,numao}@nm.cs.uec.ac.jp

あらまし　ユビキタスコンピューティングやウェアラブルコンピューティング環境下における主要なアプリケーションには, サービスのパーソナライズ化やライフログシステムがある. これらのアプリケーションの実現においては, 実世界中からユーザの「現在または過去の行動, 状態, 場所, 周囲の状況」を取得・検索可能であることが重要となってくる. 本研究では, 実世界から任意の時間, 場所における任意の種類のデータが取得可能であり, それらのデータの時系列的または空間的な変化や類似性, 傾向等を追跡・分析可能であることを実世界トレーサビリティと呼ぶ. 実世界トレーサビリティを実現する為の課題のうち, 時間・場所の一致に関する問題, データの粒度に関する問題, 問合せ記述の複雑さに関する問題について述べる. また, 実世界トレーサビリティにおける要求を記述する問合せ言語として実世界クエリ言語を設計し, その処理システムを実装する. 実世界クエリ言語と実世界クエリ言語処理システムによる, 上記問題に対するアプローチを示し, その評価を行う.

キーワード　センサネットワーク, ユビキタスコンピューティング, データストリーム処理

Query Management System for Real World Traceability

Katsuya SUZUKI† and Masayuki NUMAO†

† Graduate School of Computer Science, The University of Electro-Communications
1–5–1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan
E-mail: †{skat,numao}@nm.cs.uec.ac.jp

1. はじめに

近年, デバイスの小型化や低価格化, ネットワーク環境の普及に伴い, ユビキタスコンピューティングやウェアラブルコンピューティングが現実的なものになってきている. これらの環境下における主要アプリケーションにはサービスのパーソナライズ化やライフログシステムが挙げられる。

サービスのパーソナライズ化とは、ユーザにとってその時々に最適なサービスや、レコメンドを提供することである。サービスをパーソナライズ化して提供するためには, 現在のユーザの「行動, 状態, 場所, 時間, 周囲の状況」を分析し, 場合によってはデータベースに蓄積されたデータから類似した状況を検索し, その時どのようなサービスが提供されたかを知ることによって実現される.

ライフログシステムとは、人の日常生活における行動をデジタルデータとして記録するシステムである。ライフログシステムでは, 蓄積された膨大なログデータの中から目的の情報を効率的に検索する仕組みが必要となる. 検索を効率的に行うためには, 目的のデータが取得された際のコンテクストを用いることが有用であるとされている [5].

このように, これらのアプリケーションを構築するためには, 実世界中からユーザの「現在の行動, 状態, 場所, 周囲の状況」を取得し,「過去の行動, 状態, 場所, 周囲の状況」をデータベースから検索する機能が必要となる.

「現在の行動, 状態, 場所, 周囲の状況」は, ユーザの身につけたデバイスや周囲に設置されたデバイスから得られるデータを, 即座に, また逐次的に処理することで取得可能である. ここで処理するデータのような, デバイスから周期的に継続して送られてくるデータは, ストリームデータと呼ばれている. また,「過去の行動, 状態, 場所, 周囲の状況」は, ストリームデータの処理結果をデータベースに蓄積しておき, 必要に応じて検索することで取得することが可能である.

本研究ではこのような背景を受けて, 実世界トレーサビリティを定義する. 実世界のデータ構造をモデル化し, 実世界トレーサビリティの流れを説明する.

2. 実世界トレーサビリティ

2.1 実世界トレーサビリティの定義

実世界トレーサビリティとは「実世界から任意の時間, 場所における任意の種類のデータが取得可能であり, それらのデータの時系列的または空間的な変化や類似性, 傾向等を追跡・分析可能であること」と定義する. 例えば, 以下のような情報を取得することを考える.

- ある時刻, ある場所において観測された環境情報 (温度, 湿度等) と類似した観測値を得た場所は何処か.
- ある家庭において, ある期間中に最も電力を消費した時間帯と使われていた家電製品はなにか.
- ある特定の組合せの人物が, 同時刻に存在した場所は何処か. またその時刻, その場所の環境情報はどうなっているか.
- インフルエンザの発症者が, 潜伏期間に接触した人物とその日時, 場所が知りたい.

このような情報を取得するためには, データマイニング等の分析技術を用いて分析するのに必要十分なデータを, 実世界から取得し蓄積しておく必要がある. 実世界トレーサビリティは, ユーザの要求に応じてそれを満足するために必要十分なデータを実世界から取得・蓄積することで, データの追跡・分析が可能な状態を構築するための技術である.

2.2 実世界・実世界データ・センサデータ・リレーション

まず, 本研究で扱うデータの構造を次のように定義し概念タプルと呼ぶ.

$$< Time, Location, \{Attribute\} >$$

概念タプルは, 時刻 *Time*, 場所 *Location*, データ種の集合である {*Attribute*} によって表現される. 概念タプルを用いて, 取得するデータ構造を定義することが出来る. また,*Location* が *Time* に関して $Location = f(Time)$ という関数従属性を持ち, $Value = Attribute(Time, Location)$ として各データ種に対して実データを持つとき,

$$< Time, Location, \{Value\} >$$

これを実タプルと呼ぶ. これらを用いて実世界, 実世界データ, センサデータ, リレーションの 4 つを定義する.

実世界

{*Attribute*} が実世界中に存在する全てのデータ種からなる概念タプルによって定義され, *Time*,*Location* が連続的な値を持った実タプルを, 無限個持つ集合である. 図 1 におけるキューブが無限に大きい場合に相当する.

実世界データ

{*Attribute*} がデバイス等を用いて実際に取得可能なデータ種からなる概念タプルによって定義され, *Time*,*Location* が離散的な値を持った実タプルを, 無限個持つ集合である. 図 1 において示したデータ点のように, 実際にデバイスを用いて取得可能な点である.

センサデータ

{*Attribute*} がユーザの要求を満足する必要十分なデータ種からなる概念タプルによって定義され, *Time*,*Location* が離散的な値を持った実タプルを, 無限個持つ集合である. 図 2 におけるデータ点のように表現することができる.

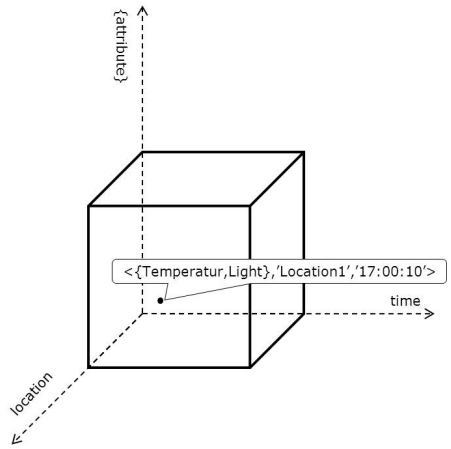

図 1　実世界と実世界データ

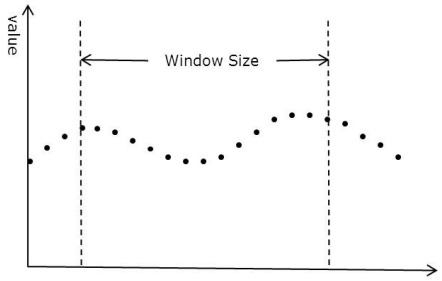

図 2　センサデータとリレーション

リレーション

{*Attribute*} がユーザの要求を満足する必要十分なデータ種からなる概念タプルによって定義され, *Time*,*Location* が離散的な値を持った実タプルを, 有限個持つ集合である. 図 2 のようにセンサデータにおいて *Time* を有限の幅で切り出した構造であり, 実タプルを蓄積するテーブルである.

これらのデータモデルを用いて, 実世界トレーサビリティは図 3 の様な流れで行われる. ユーザは実世界に対して要求を行い, 実世界から定義した実世界データと照合を行い要求が満足可能かを判定する. 満足可能であればセンサデータとしてデータを取得し, 実タプルはリレーションとして蓄積され, 要求を処理し結果を返す.

2.3 実世界トレーサビリティ実現への課題

この節では, 実世界トレーサビリティを実現するためのいくつかの課題のうち, 時間, 場所の一致に関する問題, データの粒度に関する問題, 問合せ記述の複雑さに関する問題 3 点について述べる.

2.3.1 時間, 場所の一致に関する問題

実世界トレーサビリティでは例えば,

- ある場所における同一時刻のセンサデータを結合し 1 つのセンサデータとして抽出したい.
- あるセンサデータにおいて発生した特殊なデータ（イベント）の発生時刻における, 他のセンサデータ抽出したい.

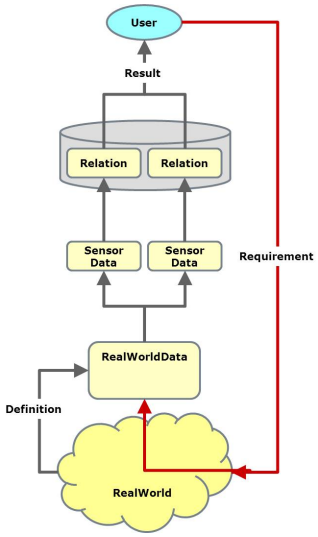

図 3　実世界トレーサビリティの流れ

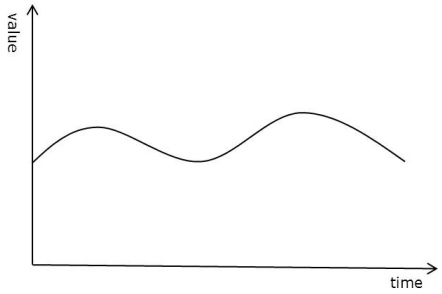

図 4　{*Attribute*} = {*Temperature*},*location* = *Location*1 における実世界のデータ列

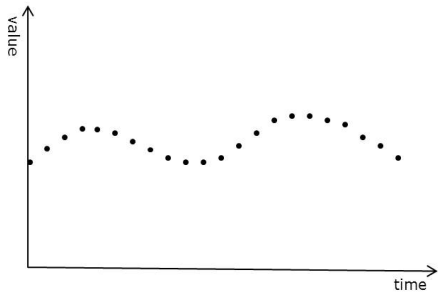

図 5　{*Attribute*} = {*Temperature*},*location* = *Location*1 におけるセンサデータのデータ列

のような場合に, いくつかのセンサデータを結合することが求められる. この時重要となるのは, 結合するセンサデータ間において *Location* や *Time* が一致していなければならないということである. 同様に蓄積されたデータに対して結合や検索を行う場合にも, そのキーとして *Location* や *Time* の一致は利用される. しかし, 異種のセンサデータ間において *Location* や *Time* が完全に一致するということは非現実的であるという問題がある.

2.3.2　データの粒度に関する問題

実世界トレーサビリティでは, 実世界から任意の時間, 場所における任意のデータを取得することが可能であるが, 図 4 の様な実タプルではなく実際には図 5 の様に時間, 場所が離散的な

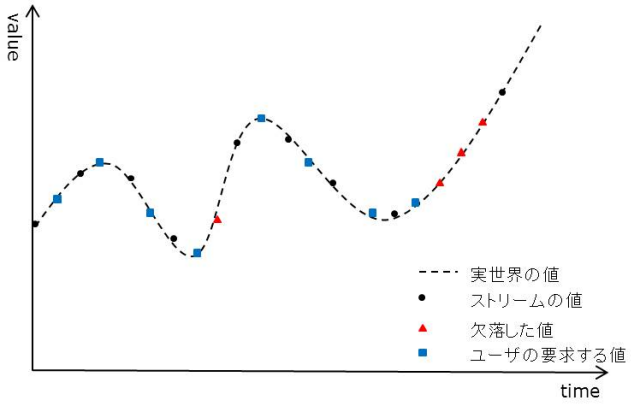

図 6　データの例

```
{CREATE SENSORDATA SensorDataName AS(
    SELECT Time,Location,Attribute,...
    FROM   RealWorld
    {WHERE conditions}
    SAMPLE PERIOD period
)}
{ON EVENT EventName AS(
   SELECT Time,Location,Attribute,...
   FROM   RealWorld
   {WHERE conditions}
   SAMPLE PERIOD period
)}
SELECT Time,Location,Attribute,...
FROM   SensorData,...,TableName,...
{WHERE conditions}
```

図 7　実世界クエリ言語の定義

値を持つ実タプルが取得される. その為, データの粒度は粗くなり, 図 6 の様にユーザの要求がその粒度より細かい場合や, データの欠落によって実データの存在しない時刻や場所のデータが要求されることも考えられるという問題がある.

2.3.3　問合せ記述の複雑さに関する問題

実世界からセンサデータとして取得したデータを RDB のテーブルに格納して蓄積することを考える. データは < *Time*, *Location*, {*Attribute*} > を 1 つのタプルとして格納されるため, テーブルのスキーマは {*Attribute*} に依存する. {*Attribute*} の異なる複数のセンサデータを格納しようとする時, カラムにヌル値を許さない場合には {*Attribute*} の内容ごとにテーブルを用意する必要がある. また, これらのデータは単一のデータベースではなく複数の分散されたデータベースに格納される場合や, その他の静的データが格納されたテーブルが存在する場合が十分に考えられる. 従って, 問合せ言語の記述はデータベースのスキーマを意識しなければならず, 非常に複雑なものとなる. さらに, 先述した時間や場所に対して誤差を許す様な問合せや, 存在しないデータに対する処理は, 通常の SQL 問合せでは可能であっても記述が煩雑または複雑である場合,SQL 問合せだけでは実現できない場合が存在する.

3.　実世界クエリ言語処理システム

3.1　実世界クエリ言語の設計

実世界トレーサビリティにおける要求を記述するための問合せ言語として, 実世界クエリ言語を図 7 の様に定義する. 実世界クエリ言語は大きく 3 つの構文から成り立っている. 以下ではそれぞれ **CREATE SENSORDATA 節**,**ON EVENT 節**,**SELECT**

```
CREATE SENSORDATA sensor AS(
    SELECT Timestamp,Location_ID,Temperature,Light,Humidity
    FROM   RealWorld
    SAMPLE PERIOD 30s
)
ON EVENT place AS(
    SELECT Timestamp,Location_ID,Tag_ID
    FROM   RealWorld
    WHERE Tag_ID='0001' OR Tag_ID='0002' OR Tag_ID='0003'
    SAMPLE PERIOD 30s
)
SELECT place.Timestamp,place.Location_ID,sensor.Temeperature,
       sensor.Light,sensor.Humidity
FROM   sensor,place
WHERE sensor.Timestamp=place.Time
    AND sensor.Location_ID=place.Location
```

図 8　実世界クエリ言語の記述例

```
ON EVENT sensor AS(
    SELECT Timestamp,Location_ID,Device_ID,Temperature,Light,Humidity
    FROM   RealWorld
    SAMPLE PERIOD 30s
)
SELECT Timestamp,Location_ID,Device_ID,Temeperature,Light,Humidity
FROM   sensor
WHERE  Temperature>25.0
```

図 9　SELECT 節でフィルタリングする場合

節と呼び説明していく. 構文中のボールド体・大文字は予約語であり, 斜体はユーザが指定する部分となっている. なお,“{ }” で囲まれた部分は省略可能である.

CREATE SENSORDATA 節

“( )” で囲まれた問合せ内容に応じて, 実世界からセンサーデータを取得する. SELECT 句に概念タプルを記述し,SAMPLE PERIOD 句で周期を記述することでセンサデータを定義する. 取得される実タプルは,*SensorDataName* で指定された名前のリレーションとして蓄積される.

ON EVENT 節

“( )” で囲まれた問合せ内容に応じて, 実世界からセンサーデータを取得する. SELECT 句に概念タプルを記述し,SAMPLE PERIOD 句で周期を記述することでセンサデータを定義する. 取得される実タプルは,*EventName* で指定された名前のリレーションとして蓄積される. また, リレーションへの新規実タプル挿入は SELECT 節を評価するトリガーとなる.

SELECT 節

実世界クエリ言語の本体となる節で, 基本的な構造は通常の SQL とほぼ同等である. ON EVENT 節が記述されている場合には, 新規実タプルが到着する度に評価される連続的問合せとして動作する. FROM 句には上記で定義したセンサデータ,DBMS 中のテーブル名が指定可能である.

実世界クエリ言語を用いて,「ある特定の人物 (Tag_ID=0001,0002,0003 を持つ人) を検知した時刻, 場所の温度, 照度, 湿度を取得する」という要求は, 図 8 の様に記述することが出来る.

実世界クエリ言語の記述力に関して, 更にいくつかの例を示す.

3.1.1　フィルタリングを用いた問合せ

図 9, 図 10 は「30 秒ごとに温度, 照度, 湿度を取得し, そのうち温度が 25 度を超えた場合に実タプルをユーザに通知する」という問合せ記述である. このように,WEHRE 句を用いることでデータのフィルタリングを行うことが出来る. この時,WHERE 句を SELECT 節で用いる場合と,ON EVENT 節で用いる場合と

```
ON EVENT sensor AS(
    SELECT Timestamp,Location_ID,Device_ID,Temperature,Light,Humidity
    FROM   RealWorld
    WHERE  Temperature>25.0
    SAMPLE PERIOD 30s
)
SELECT Timestamp,Location_ID,Device_ID,Temeperature,Light,Humidity
FROM   sensor
```

図 10　ON EVENT 節でフィルタリングする場合

```
ON EVENT sensor AS(
    SELECT Timestamp,Location_ID,Device_ID,Temperature,Light,Humidity
    FROM   RealWorld
    SAMPLE PERIOD 30s
)
SELECT sensor_hist.Timestamp,sensor_hist.Location_ID,sensor_hist.Device_ID,
       sensor_hist.Temperature,sensor_hist.Light,sensro_hist.Humidity
FROM   (SELECT * FROM sensor
        WHERE  Timestamp > SUBTIME(now(), '1:0:0')
            AND Timestamp < now()) AS sensor_hist,sensor
WHERE sensor_hist.Location_ID=sensor.Location_ID
    AND sensor_hist.Temperature=sensor.Temperature
```

図 11　新規データと履歴データとの比較

```
ON EVENT place AS(
    SELECT Timestamp,Location_ID,Tag_ID
    FROM   RealWorld
    SAMPLE PERIOD 30s
)
SELECT Timestamp,Location_ID,Human_name
FROM   place,HumanInfo
WHERE place.Tag_ID=HumanInfo.Tag_ID
```

図 12　センサデータと DBMS 中のデータの統合利用

で動作が異なる.

図 9 の様に,SELECT 節でフィルタリングを行う場合には, ON EVENT 節で定義されたセンサデータは全てリレーションとして蓄積され, その中で SELECT 節に記述した条件を満たす実タプルがユーザに通知される. CREATE SENSORDATA 節や ON EVENT 節でフィルタリグを行う場合には, 条件を満たす実タプルだけがリレーションとして蓄積され, その結果がユーザに通知される.

3.1.2　履歴データを利用する問合せ

図 11 は「30 秒ごとに温度, 照度, 湿度を取得し, 新規に追加された実タプルと, 現在から 1 時間以内の実タプルで場所と温度が一致するものを通知する.」という問合せ記述である.

SELECT 節で副問合せを用いて時間の範囲を指定することで, 最新の実タプルと履歴データとの比較を行うことが可能となる.

3.1.3　静的データを利用する問合せ

図 12 は「30 秒ごとに RFID タグを持った人を検知し, その Tag_ID を User_Name に変換して通知する」という問合せ記述である.

この様に, センサデータとして取得したデータと DBMS 中に存在するデータとを統合利用することで, デバイスから取得されたデータを加工してユーザに通知することが出来る.

3.2　実世界クエリ言語処理システムの設計

実世界クエリ言語処理システムの構成を図 13 に示す. システムは大きく 3 つのモジュールから構成されており, それぞれクエリ解析器 (**QueryAnalyzer**), コントロール生成器 (**ControlGenerator**), クエリ処理器 (**QueryManager**) から成る.

QueryAnalyzer は実世界クエリ言語の構文解析とコントロール生成有無の決定を,ControlGenerator は *Attribute*, *period* からデバイスを制御するためのコントロール生成を,QueryManager

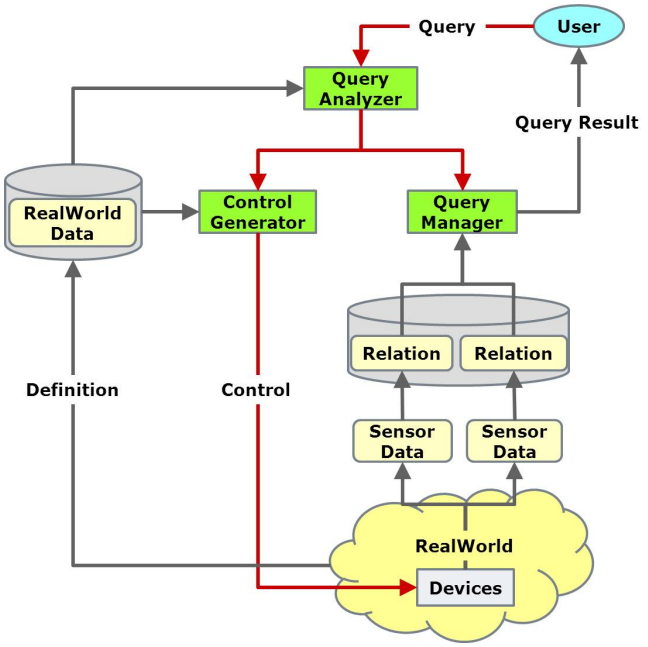

図 13　実世界クエリ言語処理システムの構成

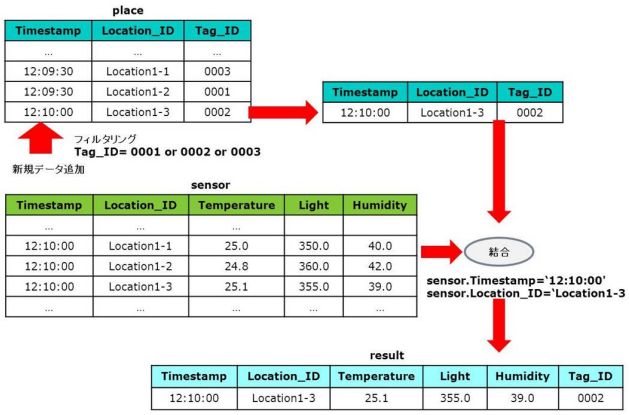

図 14　図 8 の問合せ処理例

は実世界クエリ言語の実行と, 結果の生成を行う.

例えば, 図 8 に示した実世界クエリ言語は図 14 のように処理される. place リレーションには新規に到着した実タプルのうち, フィルタリングによって $Tag\_ID = 000, 0002, 0003$ のいずれかに一致するもののみが追加されていく.place リレーションに新規に実タプルが追加されると SELECT 節の評価が開始される. 図 14 の例では,place リレーションに新規に到着した実タプルの $Timestamp = 12:00:00$ と $Location\_ID = Location1-3$ に一致する実タプルを,sensor リレーションから取得し, 結合することで問合せの処理結果を得る.

3.3　時間, 場所の一致に関する問題

WHERE 句で使用可能な演算子 ~= と関数 $NearlyEqual(Param1, Param2, ErrorRange)$ を導入することで解決している. ~= はシステムに予め登録された誤差範囲内で,$NearlyEqual(Param1, Param2, ErrorRange)$ は第一, 第二引数で指定した属性に対して $ErrorRange$ の誤差範囲内で一致とみなす.

3.4　データの粒度に関する問題

データの欠落がある場合には, 例えば SAMPLE PERIOD が 60sec であれば, 1 時間に 60 件のタプルが存在するようにデータの補完を行う. 補完の方法は, 最近傍のデータに置き換える, 前後のデータとの平均 (相加平均, 移動平均等) を取る等の方法が考えられ, どの方法が最適であるかはユーザの要求に依存する. その為, 実世界クエリ言語の発行時に指定するものとする.

また, 現在取得しているセンサデータの SAMPLE PERIOD(SP_{old}) と, 新たに与えられた問合せの SAMPLE PERIOD(SP_{new}) の差の絶対値が閾値を超える場合 ($|SP_{old} - SP_{new}| > SP_{threshold}$) には, ユーザの要求するデータの粒度が現在取得中のデータの粒度に比べて大きすぎる (小さすぎる) ため, デバイスを制御して SAMPLE PERIOD を SP_{new} に変更する.$SP_{threshold}$ の大きさについては現在検討中である.

同様に,$\{Attribute\}$ に関して, 現在取得しているセンサデータ ($\{Attribute\}_{old}$) と新たに与えられた問合せにより取得するセンサデータ ($\{Attribute\}_{new}$) が一致しない場合には, ユーザの要求に対してデータの粒度が大きすぎる (小さすぎる) ため,$\{Attribute\}_{new}$ を取得するようにデバイスを制御する. ControlGenerator によって生成するコントロールを用いた WSNs 中の無線センサデバイスの制御については, 筆者らの過去の提案 [6] を改良したシステムを利用して行う.

$Location$ を補完する方法については, 今後の検討課題である.

3.5　問合せ記述の複雑さに関する問題

実世界クエリ言語では CREATE SENSORDATA 節や ON EVENT 節を用いることにより, 自身でセンサデータを動的に定義し問合せを行うことが可能となっている. その為, 膨大なデータを持つ実世界から必要なデータのみを, 自身で定義した構造に従って取得・蓄積するため, データベース中にどのようなスキーマでデータが格納されているかを意識せずに, 直感的な記述で問合せを行うことが出来る.

4.　評　　価

実世界クエリ言語処理システムに, 時間の一致に関する問題へのアプローチとして時間に対して誤差を許す一致を利用する機能 (~= 演算子), データの粒度に関する問題へのアプローチとしてデータに欠損がある場合にはデータの補完を行う機能を実装した. この機能により,2 つの問題を解決しユーザの要求を満足する問合せ処理結果を得ることが出来ているかどうかを検証する.

また, 問合せ記述の複雑さに関する問題に対して, 実世界クエリ言語が有効であるかどうかについて考察する.

4.1　シミュレーション環境の構築

図 15 の様に建物をモデル化し, オートメーション化を想定したシミュレーション環境を構築する.

建物は,1 つ以上の部屋 ($Location$) を含む階層 ($Node$) と部屋間を結ぶ通路 ($Edge$) によって構成される. 各部屋には 1 台の RFID リーダーと, 温度, 照度, 湿度を取得可能な無線センサデバイスが設置されており, 建物を利用する人が持つ RFID タグを検知することにより, 人の所在が確認できるものとする. また,

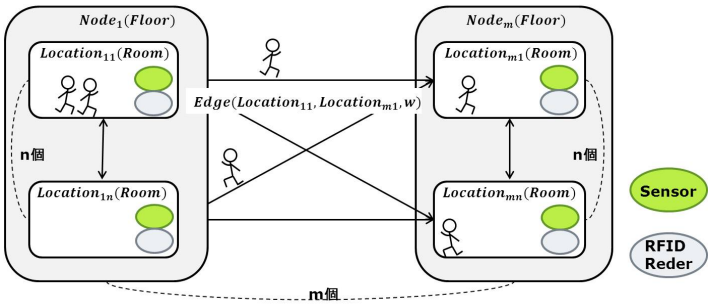

図 15　建物のモデル

表 1　静的データのスキーマ

DeviceType		
Device_Type	CHAR(20)	PK
Tag_ID	BOOLEAN	
Temperature	BOOLEAN	
Light	BOOLEAN	
Humidity	BOOLEAN	

DeviceInfo		
Device_ID	CHAR(20)	PK
Device_Type	CHAR(20)	

LocationInfo		
Location_ID	CHAR(20)	PK
Location_Name	CHAR(20)	

HumanInfo		
Tag_ID	CHAR(20)	PK
Human_Name	CHAR(20)	

表 2　動的データのスキーマ

Location		
Device_ID	CHAR(20)	PK
Timestamp	TIMESTAMP	PK
Location_ID	CHAR(20)	

Sensor	
Timestamp	TIMESTAMP
Location_ID	CHAR(20)
Temperature	DOUBLE
Light	DOUBLE
Humidity	DOUBLE

Place	
Timestamp	TIMESTAMP
Location_ID	CHAR(20)
Tag_ID	CHAR(20)

人は Edge を利用して自由に部屋間を移動できるものとし, その移動には *Edge* に与えられた重み *w* に応じた時間を要する.

このようにモデル化した環境を実現するために表 1 の静的データスキーマ, 表 2 の動的なデータスキーマを定義する.

表 1 のそれぞれのテーブルについて説明する.

DeviceType

デバイスの種類とその取得可能なデータ種の情報を保持する. *Device_Type* はデバイスの種類,*Tag_ID*,*Temperature*,*Light*, *Humidity* は概念タプルにおける {*Attribute*} の要素であり, 各データ種が取得可能かどうかを BOOLEAN 型を用いて表現する.

DeviceInfo

各デバイスがどの種類のデバイスであるかを保持する. *Device_ID* は各デバイス固有の ID,*Device_Type* はデバイスの種類を表す.

LocationInfo

場所の情報とデータを格納先のデータベースの情報を保持する.

表 3　実世界データの実装レベルの定義

RealWorldData		
Timestamp	TIMESTAMP	PK
Location_ID	CHAR(20)	PK
Device_ID	CHAR(20)	PK
Tag_ID	BOOLEAN	
Temperature	BOOLEAN	
Light	BOOLEAN	
Humidity	BOOLEAN	

Location_ID は場所固有の ID,*Location_Name* は場所名を表す.

HumanInfo

人の情報を保持する. *Tag_ID* は人が持つ RFID タグの ID,*Human_Name*) は対応する人の名前を表す.

表 2 のそれぞれのテーブルについて説明する.

Location

各デバイスがどの時刻に, どの場所にいるかを保持する. *Timestamp* は時刻,*Device_ID* はデバイス固有の ID,*Location_ID* は場所固有の ID を表す.

Sensor　CREATE SENSORDATA 節によって生成されたセンサデータをリレーションとして保持する. CREATE SENSORDATA 節で指定した名前, 概念タプルの属性のスキーマが動的に生成される. *Timestamp* は時刻,*Location_ID* は場所固有の ID,*Temeprature*, *Light*, *Humidity* はそれぞれの測定値を表す.

Place

ON EVENT 節によって生成されたセンサデータをリレーションとして保持する. ON EVENT 節で指定した名前, 概念タプルの属性のスキーマが動的に生成される. *Timestamp* は時刻,*Location_ID* は場所固有の ID,*Tag_ID* は人が持つ RFID タグの ID を表す. このテーブルに新しい実タプルが挿入される際に SELECT 節が評価される.

これらのテーブルのうち,*DeviceType*, *DeviceInfo*, *Location* を結合して得られるテーブルである, 表 3 がデータモデルにおける実世界データを実現している. 実世界データテーブルを利用することで, 各デバイスが今どこにあり, どのようなデータを取得可能であるかを知ることが出来る.

4.2　実世界クエリ言語処理システムの動作に関する評価

図 15 の環境において,4 つの *Location* を含む 4 つの *Node* からなる計 16 個の *Location* によって構成されたネットワーク中を, 20 人がランダムに移動するよう設定した.

オートメーション化を行うために, 次のようなデータを監視する問合せ要求を行うことを考える.

要求　30 秒ごとに人の所在を検知し, 人のいる部屋に関しての温度, 湿度, 照度を取得する

この要求を実現するためには,

- 30 秒ごとに人の所在を検知する
- 30 秒ごとに温度, 照度, 湿度を取得する

という 2 つのセンサデータを実世界から取得し, 人を検知したことをトリガーとしてこれら 2 つのセンサデータを時刻と場所をキーに結合するという方法で実現可能である.

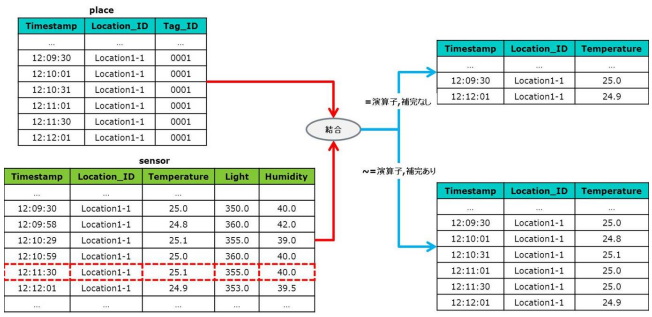

図 16　要求の処理結果例

```
CREATE SENSORDATA sensor AS(
    SELECT Timestamp,Location_ID,Temperature,Light,Humidity
    FROM   RealWorld
    SAMPLE PERIOD 30s
)
ON EVENT place AS(
   SELECT Timestamp,Location_,Tag_ID
   FROM   RealWorld
   SAMPLE PERIOD 30s
)
SELECT place.Timestamp,place.Location_ID,sensor.Temeperature,
         sensor.Light,sensor.Humidity
FROM   senor,place
WHERE sensor.Timestamp~=place.Timestamp
   AND sensor.LocationID=place.Location_ID
```

図 17　要求を満たす実世界クエリ言語

表 4　実世界クエリ言語の実行結果

	RFID タグの読み取り件数	誤差許容補完なし	誤差許容補完あり
件数	15912	3122	15912

図 16 は, この要求の処理結果が存在する時刻と場所の例を示している. place リレーションにおいて, 点線で囲まれた実タプルは通信エラー等によりデータが欠損していることを意味する.

時間の一致に関する問題, データの粒度に関する問題により,*Timestamp* の一致において = 演算子を用い, 欠損データに対する補完処理を行わない場合には, 処理結果は 2 件しか得ることが出来ない. 要求の性質から人を検知した時刻, 場所における温度, 照度, 湿度の情報は必ず存在しなければならないため, これでは要求を満足する結果とは言えない.

そこで, 時刻の一致には　= 演算子を用い, データが欠損している場合には最近傍のデータで補完する機能を実装し, 図 17 の実世界クエリ言語による問合せを実行した.

問合せ処理の結果を表 4 に示す. 誤差の許容とデータの補完を用いない場合には, 結果の件数が RFID タグの読み取り件数に比べて著しく少なく, 時間の一致に関する問題とデータの粒度に関する問題の影響が見て取れる. 誤差の許容とデータの補完を用いた場合には, 結果の件数が RFID タグの読み取り件数に一致し, 要求を満足する問合せ結果を得ることが出来ている.

これにより, 時間の一致に関する問題とデータの粒度に関する問題に対するアプローチとして, 誤差を許した一致処理とデータの補完を用いることが有効であるといえる.

4.3　実世界クエリ言語に関する評価

実世界クエリ言語は,3.1 節の図 7 で示した構文を持つ問合せ言語である. CREATE SENSORDATA 節,ON EVENT 節を用いることでユーザがセンサデータを動的に定義することが可能であり, SQL ライクな構文を持つ. センサデータを定義するとは, 言い換えれば仮想的なデバイスを定義していることを意味しており, 各データ種がどのデバイスによって取得されるのかという, 物理的な制約をユーザは意識する必要が無くなる. また, データを蓄積するためのリレーションがユーザの定義により動的に生成されるため, データベースのデータスキーマを意識することなく直感的な記述によって, 実世界からデータを取り出すことが出来るという特徴を持つ.

さらに,ON EVENT 節を用いることで連続的問合せを行うこと,DBMS 中のテーブルとの統合利用することが可能であるため, 問合せ言語としての要求記述力は通常の SQL に比べて増したと言える.

一方で, 利用するセンサデータを問い合わせ時に記述しなければならず, 記述量が増してしまう場合も存在するため, 問合せ記述の複雑さに関する問題への有効性について今後も検討していく必要がある.

5.　関 連 研 究

TinyDB [2] や,Cougar [3] は WSNs を対象としたストリームデータ処理エンジンであると見なすことが出来る. SQL ライクな問合せ言語を用いて, 必要な種類のデータを必要なタイミングで取得する事は, 本研究における実世界データからセンサデータを取得することに相当する. しかし, これらの研究では取得したデータを蓄積することや, 新規に到着したデータと蓄積されたデータとの統合利用は行われていない.

STREAM [1] や SASE+ [4] の様なストリーム処理エンジンは, 新規に到着したデータに対して, 連続的問合せを用いて即座に処理することに重点が置かれている. その為, 新規に到着したデータをデータベースに蓄積することや, 蓄積したデータとの統合利用に関しては不向きであると言える.

また, 従来の DBMS [10] [11] は新規に到着したデータを蓄積することや蓄積されたデータに対する処理は可能であるが, 新規に到着したデータに対して連続的な処理を行うことには不向きであるとされていた. しかし, トリガー機能を用いることにより能動的な処理が可能となった. 本研究における ON EVENT 文を用いた問合せは, このトリガーを定義することに相当する.

StreamSpinner [9] [8] は,SQL ライクな問合せ言語を用いて複数のデータストリームを統合利用することや, 新規に到着したデータを蓄積し, 蓄積したデータとストリームデータとの統合利用を行うことが出来る. 本研究では, 問合せ内容に応じてストリームデータを生成するデバイスを制御することにより, デバイスの取得するデータ種や, 取得タイミングを動的に変更可能である点が異なる. また, ストリームデータや蓄積されたデータの結合処理において, 時間や場所の完全一致が非現実的である事を考慮し, 一致処理において誤差を許容する仕組み, データの欠損などの理由によりユーザの要求するデータの粒度を満足できない場合にデータの補完を行うことで擬似的なデータを通知する機能を提供している.

実世界トレーサビリティを実現するためには, 実世界に対する様々なユーザの問合せを要求を満足可能にする必要がある.

その為には, 実世界という稠密なデータセットを蓄積しておかなければならないが, これは非現実的である. そこで本研究では, 実世界クエリ言語を用いた問合せを行い, 実世界中に点在するデバイスを制御するためのコントロールを生成, 送信することにより, 問合せ内容を満足する為に必要十分なデータのみをセンサデータとして収集するように, デバイスを動的に制御する. これにより, 既存のストリーム処理エンジンと異なり, 扱うストリームをユーザの要求に応じて動的に変更することが可能となっている点が異なる.

6. まとめと今後の課題

本研究では実世界トレーサビリティを定義し, その為のデータモデルとして, 実世界, 実世界データ, センサデータ, リレーションのモデル化を行った. また, 実世界トレーサビリティを実現する上で課題となるもののうち, 時間, 場所の一致に関する問題, データの粒度に関する問題, 問合せ記述の複雑さに関する問題について述べた. 実世界トレーサビリティにおいて要求を記述するための問合せ言語として実世界クエリ言語を設計し, その処理システムの設計と実装を行い, シミュレーション実験により誤差を許す一致機能, 欠損データの補完機能が有効であることを示した.

今後の課題としては, 複数の異なる問合せ処理を並列に行う為に, 問合せの合成や書き換えによる問合せの最適化, 実行プランの最適化を行う仕組みを実装こと, 実環境下において本研究の実装を行い評価を行うことが挙げられる.

文　　献

[1] Arvind Arasu, Brain Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom.“ STREAM: The Stanford Stream Data Manager ”. In Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data, pp.665, June 2003. Demo description (2003).

[2] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong.“ TinyDB: an acquisitional query processing system for sensor networks ”. ACM Transactions on Database Systems (TODS), 30, 1, pp.122.173 (2005).

[3] Alan Demers, Johannes Gehrke, Rajmohan Rajaraman, Niki Trigoni, and Yong Yao.“ The Cougar project: a work-in-progress report. ” ACM SIGMOD Record, Vol.32,No.3,pp.53-59 (2003).

[4] Yanlei Diao, Neil Immerman, and Daniel Gyllstrom.“ SASE+: An Agile Language for Kleene Closure over Event Streams ”. In Technical Report 07-03, UMass Amherst (2007).

[5] 堀 鉄郎, 河崎 晋也, 石川 尊之, 相澤 清晴. “ ライフログ応用に向けたコンテキストに基づく映像・データ検索 ”, 電子情報通信学会技術研究報告. MVE, マルチメディア・仮想環境基礎 103(745), pp.55-60 (2004).

[6] 鈴木克弥, 沼尾雅之. “ センサネットワークにおける問い合わせ言語を用いたデータ収集効率化手法の提案 ”. 第 23 回人工知能学会全国大会 (2009).

[7] Shili Xiang, Hock Beng Lim, Kian-Lee Tan, Yongluan Zhou.“ Two-tier multiple query optimization for sensor network ”. In: Proc. of ICDCS, p. 39 (2007).

[8] 山田 真一, 渡辺 陽介, 北川 博之.“ ストリーム統合システムを用いた実世界情報の統合利用 ”. 第 16 回データ工学ワークショップ (2005).

[9] StreamSpinner. http://www.streamspinner.org/

[10] MySQL. http://www.mysql.com/

[11] PostgreSQL. http://www.postgresql.org/