Vol.2022-SE-210 No.7
2022/3/11

# 永続時変値の分散化に向けた基盤システムの試作

上野 颯太[1,a)]　紙名 哲生[1,b)]

**概要**：リアクティブプログラミング（RP）における時変値では，変数への明示的な再代入を行うかわりに，変数の値が時間とともに変化する．さらに Java に基づく RP 言語 SignalJ においては，時変値の値変化の履歴を時系列データとみなし，その値を指定された時刻の関数とする永続時変値の機構が提供される．SignalJ は元々ローカルなシステムを対象としてきたが，永続時変値の実体は言語処理系の外部にある時系列データベースであり，オープンなリアクティブシステムの基盤となりうるものである．本研究ではこの特徴を生かした，永続時変値のための分散計算基盤の実現方法を示す．またそれを用いたソフトウェア開発を現実のものにするために解決すべき様々な問題について議論する．

**キーワード**：リアクティブプログラミング，時系列データベース，名前解決，SignalJ

# An implementation of infrastructure system for distributed persistent signals

Sota Ueno[1,a)]　Tetsuo Kamina[1,b)]

***Abstract:*** A signal in reactive programming (RP) is considered as a function of time and its value is updated automatically without performing explicit assignments. Furthermore, the Java-based RP language SignalJ provides persistent signals, where the update history of a signal is considered time-series data, and the value of the signal is considered as a function of a specified (possibly past) time. SignalJ have been targeting to local systems; however, the entity of a persistent signal exists in the time-series database, which exists in the outside of the language and can be a basis of open reactive systems. In this paper, we propose an implementation of infrastructure system for open and distributed persistent signals by utilizing such a characteristic. We also discuss several issues on realizing software development based on this infrastructure.

***Keywords:*** Reactive programming, Time-series databases, Name resolution, SignalJ

## 1. はじめに

現在，Web アプリケーションやモバイルアプリケーションにおいて，連続して入力を受け取り，その度に応答を返すリアクティブなシステムが注目を集めている．現在では多くのアプリケーションにリアクティブな性質が取り入れられている．例として，Microsoft Excel などの表計算ソフト，ロボットの制御プログラム，ユーザーインターフェースなどがあげられる．これらはいずれも，ある入力に対してなんらかの処理を行い，その入力に依存した応答を返すことを，起動中ずっと繰り返すものである．従来の命令的なプログラミング手法を用いると，この振る舞いは入力の通知，値の更新を主にイベント処理などの仕組みを用いてプログラマが明示的に実装する必要がある．

一方，連続して得られる入力値とそれに依存する値を「時間変化する値（時変値）」として抽象化し，入力から応答に至るデータフローを宣言的に記述できるリアクティブプログラミング（RP）言語が数多く知られている．それらの中でも，Java に基づく RP 言語 SignalJ[6] においては．時変値の値変化の履歴を時系列データとみなし，その値を

[1] 大分大学理工学部共創理工学科知能情報システムコース
Division of Computer Science and Intelligent Systems, Oita University, Dannoharu 700, Oita 870-1192, Japan
[a)] v1858208@oita-u.ac.jp
[b)] kamina@acm.org

ⓒ 2022 Information Processing Society of Japan

指定された時刻の関数とする永続時変値の機構が提供される [7], [8].

SignalJ は元々ローカルなシステムを対象としているが，永続時変値の実体は言語処理系の外部にある時系列データベースであり，オープンなリアクティブシステムの基盤となりうるものである．しかし，SignalJ では設定ファイルに紐づいている単一のデータベースに格納されている時変値のみの参照を行っているため，そのような基盤の実現を可能にしていない．著者らは，そのような基盤を実現することで，オープンな時系列データ，例えば気象情報などの時系列データを SignalJ の時変値として利用することができるようになり，広域ネットワーク上に分散する各種の時系列データを宣言的なデータフローとして記述できるようになると考えた．

本研究の目的は，SignalJ を分散化したデータベースに紐づいている時変値に対応させ，オープンなリアクティブシステムの基盤を作ることである．SignalJ では，永続時変値は唯一の名前（id）を持ち，その名前で「永続時変値の実体」となる時系列データベースのテーブルと紐づけられている．そこで，時系列データベースの一つである TimescaleDB を分散化させたデータベースに時変値を格納し，id に基づく名前解決テーブルを用いることにより複数のデータベースに分散した時変値を参照させる．

この仕組みの実現可能性を確認するため，時系列データベースを著者らが所属する研究室内のサーバとクラウド上にそれぞれ実装し，それらにアクセスする永続時変値を持つアプリケーションを開発し，動作を確認した．具体的には研究室内サーバの時系列データベースに温かい飲み物と冷たい飲み物の販売数，クラウド上の時系列データベースに気象情報（最高気温，平均気温，最低気温）を格納し，それらの時系列データを時変値として参照してそれらの間の相関係数を計算するプログラムを作成する．また本研究のプログラムの実装においてオープンなリアクティブシステムの実現可能性と問題点について考察する．

本論文の構成は以下のとおりである．2 節で，本研究の準備として必要な要素技術について述べる．3 節で，本研究での分散化されたデータベースへのアクセスに向けた概要と機能について述べる．4 節で，本研究での分散化したデータベースへのアクセスの実現可能性を検証するために実装したシステムとその実行に対する考察を述べる．5 節で，本研究における今後の課題を考察する．6 節で本研究のまとめを述べる．

## 2. 要素技術

### 2.1 リアクティブプログラミング（RP）

RP とは，時間変化する時変値同士の依存関係や計算方法を宣言的に記述するプログラミング手法である．RP では，データフローの入り口にあたる時変値が変化したタイミングでその時変値に依存する他の全ての時変値の値を再計算するので，時変値同士の関係を宣言するだけでプログラムの振る舞いを記述することができ．コードが簡略化される．

時変値とはプログラムの変数の一種であるが，通常の変数とは異なり変数への明示的な再代入を行うかわりに，変数の値が時間とともに変化する．例として SignalJ による時変値の宣言を以下に示す．

```
class Vehicle {
    signal double x, y;
    signal double dx = x.lastDiff(1);
    signal double dy = y.lastDiff(1);
    signal double v = dx.distance(dy);
}
```

これは SignalJ で宣言した車両（`Vehicle`）クラスである．SignalJ では時変値を宣言する際，修飾子 `signal` を用いる．上の例では，`Vehicle` クラスに車の x 座標，y 座標，加速度，速度がそれぞれ時変値 `x`，`y`，`dx`，`dy`，`v` を用いて宣言されている．加速度は x 座標，y 座標を用いて計算するメソッドにより求められ，速度は加速度を用いて計算するメソッドにより求められている．x 座標，y 座標が更新されると加速度も自動的に再計算され，新しい値に更新される．速度も同じく x 座標，y 座標が更新され加速度が再計算されたタイミングで新しい値に更新される．つまり，`signal` では入力と出力の依存関係を定義しておけば自動的に値の更新が行われる．これにより，プログラマはイベントの状態管理や出力の更新などの記述から解放され，入力と出力の関係の定義に集中できる．シンプルで個人差の少ない記述が行えるため，開発効率，保守性の向上が見込めるといった点が時変値を用いるメリットとなっている．

シグナルはもともと Fran[3] や FrTime[1] のような関数リアクティブプログラミング（FRP）言語で議論されてきた言語要素であるが，今では上述した SignalJ や REScala[11], PuPPy[12] のように，命令的な言語要素を含む既存のプログラミング言語を拡張した実装も存在する．後者では，代入演算子などを用いて命令的にシグナルの値を更新することも可能である[*1]．

### 2.2 永続時変値

永続時変値とは時変値の値変化の履歴を含めて抽象化した時変値のことで，内部的には，履歴の時刻印をキーにした時系列データベースを用いて実装される．これは，気象情報などの時間変化する永続データに問い合わせができ，過去のデータへの遡及を可能にしている．例として先ほどの `Vehicle` クラスを永続時変値を用いて宣言したクラスを

[*1] 更新できるのは通常，他のシグナルに依存しない，データフローの入り口に位置するシグナルのみである．

以下に示す．

```
signal class Vehicle {
    persistent signal double x, y;
    signal double dx = x.lastDiff(1);
    signal double dy = y.lastDiff(1);
    signal double v =  dx.distance(dy);
    Vehicle(String id){:}
}
```

先ほどの時変値の例で挙げた Vehicle クラスと同様に各車両は状態として現在の座標値を表す x，y，加速度 dx，dy，速度 v を持つ．これらはどれも時変値である．さらにここでは，座標値 x，y は永続時変値であることを示す修飾子 persistent が付けられている．ここでは persistent と連携する時変値も永続時変値として扱われるため，dx，dy，v も永続時変値である．各永続時変値に，Vehicle クラスをインスタンス化する際に明示的な識別子が与えられ，その識別子と紐づいている．車両の識別子を与えるパラメータとして id を用いているのでコンストラクタ中の仮引数 id は必須である．この例に限らず，signal と修飾されたクラス（以下，シグナルクラスと呼ぶ）は persistent と修飾された時変値をまとめるためのものであり，必ずこの仮引数を要求する．

永続時変値は，過去への遡及を行う計算をする際に便利である．例えば，ある車両が過去に危険な速度を出しており，過去の速度や座標を求める必要がある場合がある．SignalJ では指定した時刻における過去の速度を求めるのに，以下のように snapshot メソッドを用いる．これはシグナルクラスが暗黙的に持つメソッドである．

```
double ve = new Vehicle(“oita1234”).
    snapshot(“2020-01-01T18:10:00”).v;
```

車両クラスのインスタンス生成に識別子として”oita1234”を与えることにより，その識別子と紐づいた時変値（x，y，dx，dy，v）を得る．次に snapshot メソッドを用いて時間を遡り，フィールド v にアクセスすることでその時点での速度を得る．このように永続時変値では過去の値に対する遡及を可能にしている．上で生成されたインスタンスが消滅したりアプリケーションそのものが停止したとしても，時変値が持つ履歴は明示的に破壊されない限りデータベースに残り続ける．このように，永続時変値では通常の時変値とは異なるライフサイクル管理の機構が用意されている．

バックエンドとなる時系列データベースには TimescaleDB が用いられている．これは，IoT 機器のセンサやスマートメータが生成するデータのように，時系列に沿って発生するデータを扱うのに適した時系列データベースの機能を，PostgreSQL に追加するための拡張モジュールである．時系列データベースに関する問い合わせ文中で使用する組み込み関数が多数用意されてあり，従来通りの SQL の構文を用いた問い合わせが可能である．PostgreSQL に基づいているので，時系列以外の機能もバランスよく実装されている．TimascaleDB では，時系列データを格納したテーブル（Hypertable）を PostgreSQL のパーティショニング機能を用いて日時間隔別の小さなチャンクに分割するという実装方法がとられている．

### 2.3 分散 RP の先行研究

分散 RP に関する先行研究として，グリッチの回避手法に関する研究があげられる．グリッチとは，値更新時に起こる一時的な不整合のことである．一般的には，こうした不整合はクロックを用いたり [4] 各時変値の計算の順序を正しく決めれば回避できる [9]．しかし，それぞれの時変値が異なる計算ノードで計算される分散環境においてはこの回避は容易ではない．まず，ネットワークの遅延や障害の影響を考慮に入れる必要がある．また，時変値の計算の順序を誰がどのように決定するのかという問題がある．閉じた一つのプログラムにおいては，それはコンパイラが決定する．一方分散 RP 言語においては，特別なコーディネータが決定を行うか（コーディネータで障害が起きた際の影響は大きい）[2]，あるいはノード同士が情報を送りあって計算の順序を決定するような分散アルゴリズムが考えられている [10]．一方，本研究のように，分散された時系列データベースを RP 言語の基盤とする取り組みはこれまでに知られていない．

## 3. SignalJ でのバックエンドデータベースの分散化

### 3.1 時系列 DB を分散化することの意義

現在 SignalJ では永続時変値を保存するデータベースは分散化されておらず，設定ファイルで指定したデータベースに時変値を保存している．SignalJ は元々ローカルなシステムを対象としてきたが，永続時変値の実体は言語処理系の外部にある時系列データベースであり，オープンなリアクティブシステムの基盤となりうるものである．時系列データベースを分散化させることにより，既存のオープンな時系列データベースを SignalJ で利用しそれを用いたソフトウェア開発することができると考えられる．また，現在の SignalJ は単一のデータベースに時変値を保存していることにより，データベースに保存できる時変値のキャパシティには限度がある．さらにデータベースに多くの時変値を保存することで応答性能は低下すると考えられる．本研究では時系列データベースを分散化することによって，既存のオープンな時系列データベースの利用や，通信負荷のパフォーマンスの向上を目指す．

ⓒ 2022 Information Processing Society of Japan

```
signal class HotWater{
    persistent signal double hotwater;
    double [] hwdeviation = new double [7];
    double sdeviation;
    Timestamp ts;
    signal double hwater7days =
        hotwater.within( ts, "7 days");
    double hwave = hwater7days.avg();
    public HotWater (String id,  Timestamp ts) {
        this.ts =ts;
    }
    ...
    public void calculation() {
        double [] value = new double[7];
        double [] distributed = new double[7];
        ...
        for(int i=0; i<7; i++) {
            ...
            this.snapshot(new Timestamp(...));
            value[i] =  hotwater;
        }
        for(int i=0; i<7; i++) {
            hwdeviation[i] = value[i] - hwave ;
        }
        ...
    }
}
```

**図 1** HotWater クラス

### 3.2 サンプルプログラム

分散化された環境における SignalJ プログラムの実行を確認するためにサンプルプログラムを準備する．サンプルプログラムでは温かい飲み物の販売数の時系列データと冷たい飲み物の販売数の時系列データがあることを想定し，それとは独立した時系列データとして気象情報を用いる．サンプルプログラムでは，これらはそれぞれ異なる時系列データベースに分散配置されて格納されていることを想定するが，それらを SignalJ の時変値として参照する際は，id による指定のみで格納場所については意識しない．このプログラムは，気象情報と温かい飲み物，冷たい飲み物の販売数の相関を計算する．

図 1 に，本研究で用いるサンプルプログラムを示す．このプログラムは温かい飲み物の販売数を時変値として参照するクラス HotWater，冷たい飲み物の販売数を時変値として表現するクラス ColdWater，気象情報を時変値として表現するクラス Weather，そしてこれらの相関係数を求めるクラス Correlation で構成されている．

HotWater は，永続時変値 hotwater を持っている．相関を求めるためには過去の値を参照する必要がある．そこで SignalJ がもつ within(Timestamp ts, String duration) メソッドを用いる．このメソッドは時刻印から引数 duration で指定された期間の永続時変値を返す．つまり，hwater7days は ts（within で指定した時刻印）から過去 7 日間の hotwater の値を履歴を持つ永続時変値である．double 型の変数 hwave も同じく SignalJ が持つ avg() メソッドを用いて温かい飲み物の販売数の 7 日間の平均を求めている．avg() メソッドはレシーバ永続時変値の平均を求めるメソッドである．コンストラクタの引数 id はインスタンス作成時に必要とされる識別子である．また，double 型の配列 hwdeviation と double 型の sdeviation がフィールドとして宣言されているが，これらは 1 日ごとの温かい飲み物の販売数の偏差と標準偏差で，calculation メソッドで計算している．calculation メソッドは偏差と分散を計算するメソッドである．Timestamp 型のオブジェクトを用いてシグナルクラスが暗黙的に持つメソッド snapshot メソッドにより自分自身の時刻印を更新することにより，日ごとの値を配列 value に代入している．また，配列 value の値を用いて日ごとの偏差，日ごとの分散，標準偏差を計算している．

```
public class Correlation {
    public static void main(String[] args) {
        String datetime = "2022-01-01 23:59:59";
        Timestamp timestamp =
            Timestamp.valueOf(datetime);
        HotWater  h = new HotWater ("000a",timestamp);
        ColdWater c = new ColdWater("000b",timestamp);
        Weather   w = new Weather("000c",timestamp);
        h.calculation()  ;
        c.calculation();
        w.calculation()  ;
        System.out.println("Coefficient: "+
            coefficient(h.hwdeviation,
                        h.sdeviation,
                        w.atemdeviation,
                        w.atemsdeviation));
    }
    ...
}
```

**図 2** Correlation クラス

ColdWater クラス，Weather クラスは HotWater と大きな違いはないため省略する．Weather クラスは HotWater クラスや ColdWater クラスと異なり永続時変値として日ごとの最高気温，最低気温，平均気温を保持しており，それぞれの日ごとの偏差と標準偏差を求めている．

Correlation クラスでは，HotWater，ColdWater，Weather を，それぞれに固有の識別子と指定された Timestamp 型の時刻を引数としてコンストラクタを呼ぶことにより，それらのインスタンスを作る．それぞれのインスタンスに対して calculation メソッドを呼び出し，それぞれの時変値の分散，標準偏差を計算した後 Correlation

ⓒ 2022 Information Processing Society of Japan

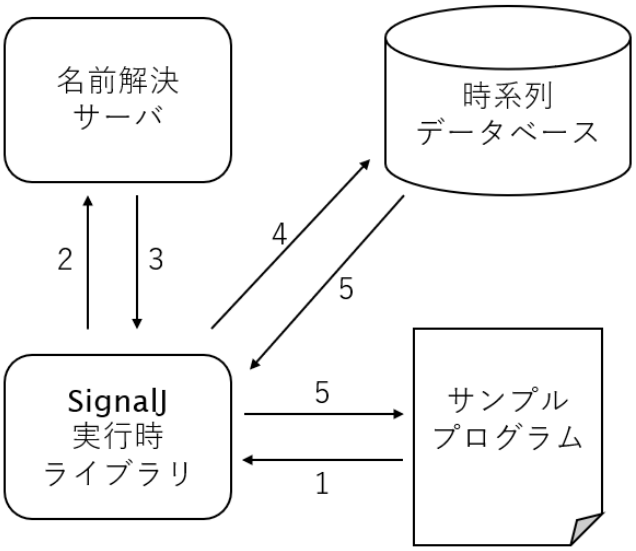


**図 3** 分散計算基盤のアーキテクチャ

クラス内で用意された `coefficient` メソッドを呼び出すことでそれぞれの相関係数を出力する．`coefficient` メソッドは各クラスの標準偏差と分散を引数として受け取り相関係数を計算するメソッドである．日ごとの分散を用いて共分散を計算し，共分散と標準偏差を使用し相関係数を求め戻り値としている．

### 3.3 アプローチ

現在 SignalJ では設定ファイルで時系列データベースと時変値を紐づけており，ローカルなシステムを対象としている．外部のオープンな時系列データベースを対象とするために，自動でデータベースと時変値を紐づけるシステムを構築する必要がある．ここで，本研究では DNS のような名前解決機構を SignalJ ライブラリに追加する．

図 3 に，サンプルプログラムを用いて，本研究で提案する永続時変値のための分散計算基盤のアーキテクチャを示す．サンプルプログラムから参照される SignalJ 実行時ライブラリは名前解決サーバに接続し，id による問い合わせを行う．時系列データベースは分散化されており，具体的な時系列データが格納される．以下時変値参照までの流れを示す．

(1) サンプルプログラムが実行されると固有の識別子 id を用いて永続時変値を参照するインスタンスが作られる．これらのインスタンスは SignalJ 実行時ライブラリによって実現され，サンプルプログラム内で指定された id を参照する．
(2) SignalJ 実行時ライブラリは名前解決サーバに id を用いて時系列データベースにアクセスするための情報について問い合わせる．
(3) 問い合わせを受け付けた名前解決サーバはサーバ内にある名前解決テーブルを参照し id に紐づいている時系列データベースの url，ユーザ名，パスワードを SignalJ 実行時ライブラリに返す．
(4) SignalJ 実行時ライブラリは返された url，ユーザ名，パスワードを用いて時系列データベースにアクセスする．
(5) サンプルプログラムは時系列データベースに格納されている時変値の問い合わせを行う．

**表 1** 名前解決テーブルの内容

| id | url | user | password |
|---|---|---|---|
| correlation_hotwater_000a | ... | ... | ... |
| correlation_coldwater_000b | ... | ... | ... |
| correlation_weather_000c | ... | ... | ... |

### 3.4 名前解決サーバの仕組み

本研究では，名前解決テーブルは PostgreSQL を用いて試作した．名前解決テーブルでは，SignalJ 実行時ライブラリから id による問い合わせが行われた場合，id に紐づいた url，ユーザ名，パスワードを返している．id には，クラスのフルパス名で修飾された id が格納されてい my る．これがそのまま id に紐づけられた時系列データベースのテーブル名になっている．表 1 に PostgreSQL により作成したサンプルプログラムが用いる名前解決テーブルを示す．

### 3.5 SignalJ 実行時ライブラリの拡張

本研究では既存の SignalJ 実行時ライブラリの `TimescaleSignal` クラス内の `connectDB` メソッドに名前解決を行う機能を追加している．これは永続時変値が，自身の値更新履歴を保存する時系列データベースに接続する際に呼ばれるメソッドであり，従来は java.properties ファイルで設定された時系列データベースの接続情報（url，ユーザ名，パスワード）を参照していた．本研究では，java.properties ファイルからそれらの設定情報を参照するのではなく，connectDB メソッドから id を用いて名前解決サーバに問い合わせを行い，名前解決サーバから返された url，ユーザ名，パスワードを用いて分散化された時系列データベースにアクセスする機能を追加した．

## 4. 検証

本章では分散化したデータベースへのアクセスの実現可能性を検証するために実装したシステムとその結果に対する考察を述べる．

### 4.1 検証概要

本検証は分散化した時系列データベースの時変値の参照，名前解決手法の実現可能性と今後の課題を調査することが目的である．本実装では，3 節で示した時系列データベースの分散化に対応した SignalJ 実行時ライブラリを開発し

```
public class CreateTables {
    public static void main(String[] args) {
        String datetime = "2022-01-01 23:59:59";
        Timestamp timestamp =
            Timestamp.valueOf(datetime);
        HotWater h = new HotWater("000a", timestamp);
        ColdWater c = new ColdWater("000b", timestamp);
        Weather w = new Weather("000c", timestamp);
    }
}
```

**図 4** 永続時変値のインスタンス化を行うプログラム

たうえで，それを用いてサンプルプログラムを動かし，動作を確認した．まず，研究室内のサーバとクラウド上にそれぞれ TimescaleDB を準備し，サンプルプログラムで用いる暖かい飲み物と冷たい 飲み物それぞれの販売数を格納するためのテーブルを研究室内のサーバに，気象情報を格納するためのテーブルをクラウド上に用意する．次に，これら時系列データベースに気象情報，温かい飲み物の販売数，冷たい飲み物の販売数のデータを挿入する．そして，名前解決テーブルにも予め，id とそれぞれの時系列データベースへの接続情報を準備しておく．その環境下でサンプルプログラムを SignalJ コンパイラでコンパイルし，実行した．

### 4.2 検証手順

分散化した時系列データベースに値を挿入する．今回は研究室内のサーバとクラウド内の二つの時系列データベースを利用した．時系列データは，Microsoft Excel などの形式で提供されることを想定しているが，今回はテーブルを自動生成した後，データを手動で投入する方法をとった．まず，時変値を作るクラス `CreateTables` を作成する（図 4）．SignalJ では `signal` と修飾されたクラスのインスタンス化を行った際に対応するテーブルがなければ自動的に作成されるため，インスタンス化だけを行うプログラムとして作成し，実行した．

`CreateTables` では，`HotWater`，`ColdWater`，`Weather` を名前解決に使用するデータベース固有の識別 id と任意の指定した `Timestamp` 型の時刻を引数としインスタンス化を行う．TimescaleDB に値の挿入を行うにはテーブルを作成し，両方のサーバともにあらかじめ 7 日間の販売数を書き示した CSV ファイルから psql のコピーコマンドを用いてコピーを行う．研究室内のサーバとクラウド上のサーバはともに TimescaleDB を用いているので同様の操作を行い，研究室内のサーバには温かい飲み物の販売数のテーブルと冷たい飲み物の販売数のデータを投入し，クラウド上のサーバには気象情報のデータを投入した．

図 5，6，7 にそれぞれ温かい飲み物，冷たい飲み物，気象情報を投入したテーブルを示す．

```
persistent-signals=> select * from correlation_hotwater_000a ;
 id |          time          | hotwater
----+------------------------+----------
  1 | 2022-01-01 23:59:59-05 |       18
  2 | 2022-01-02 23:59:59-05 |       14
  3 | 2022-01-03 23:59:59-05 |       10
  4 | 2022-01-04 23:59:59-05 |       16
  5 | 2022-01-05 23:59:59-05 |       17
  6 | 2022-01-06 23:59:59-05 |       10
  7 | 2022-01-07 23:59:59-05 |       13
(7 行)
```

**図 5** 温かい飲み物テーブル

```
persistent-signals=> select * from correlation_coldwater_000b ;
 id |          time          | coldwater
----+------------------------+-----------
  1 | 2022-01-01 23:59:59-05 |         2
  2 | 2022-01-02 23:59:59-05 |        10
  3 | 2022-01-03 23:59:59-05 |         8
  4 | 2022-01-04 23:59:59-05 |        10
  5 | 2022-01-05 23:59:59-05 |         7
  6 | 2022-01-06 23:59:59-05 |         8
  7 | 2022-01-07 23:59:59-05 |         9
(7 行)
```

**図 6** 冷たい飲み物テーブル

```
persistent-signals=> select * from correlation_weather_000c;
 id |          time          | htempreture | atempreture | ltempreture
----+------------------------+-------------+-------------+-------------
  1 | 2022-01-01 23:59:59+00 |           9 |         5.5 |           0
  2 | 2022-01-02 23:59:59+00 |          12 |           6 |           0
  3 | 2022-01-03 23:59:59+00 |          10 |           5 |           0
  4 | 2022-01-04 23:59:59+00 |          12 |         7.3 |           1
  5 | 2022-01-05 23:59:59+00 |          11 |         6.2 |           1
  6 | 2022-01-06 23:59:59+00 |          12 |         6.6 |           2
  7 | 2022-01-07 23:59:59+00 |          12 |         6.2 |           2
(7 行)
```

**図 7** 気象情報テーブル

この条件のものでサンプルプログラムを実行し，分散化した時系列データベースの時変値の値を参照して，温かい飲み物の販売数，冷たい飲み物の販売数と気象情報をもとに計算した相関係数をそれぞれ計算し出力した．

### 4.3 考察

本研究により，分散化された時系列データベースに基づく時変値を用いて相関係数を計算することに成功した．もっとも今回の研究では既存のオープンな時系列データベースがあることが前提であり，また時系列データに変更があった場合や，そもそもどのようなデータが利用可能なのかを考える必要がある．さらに，今回の名前解決テーブルは予め id と url が紐づけられているという前提であるが，実際は名前解決サーバへ自動的に id と url を登録する仕組みが必要である．これらの課題は 5 節で詳細に扱う．いずれにせよ，本研究は分散化を考えずローカルな仕組みを対象とした SignalJ に，オープンなリアクティブシステムの基盤を提供する最初の取組みである．

また，今回は名前解決を PostgreSQL の名前解決テーブルを用いて行ったが，これをオープンなリアクティブ基盤とすることを考えると，id の数は非常に大量になることが予想されるため，このままではスケールしない．そこで，

分散化された KeyValueStore を用いた名前解決を考えている．現在利用可能な KeyValueStore は多くあり実際に複数のサーバにデータを分散させて利用する分散データベースストアとしての使用が増えている．名前解決は単純な検索なので KeyValueStore による実現に向いており，またここがボトルネックになると全体のパフォーマンス低下が懸念されるため，分散化してスケールさせる仕組みが必要である．

## 5. 議論

本研究における今後の課題を考察する．

### 5.1 id 解決における登録のタイミング

本研究では，名前解決サーバには予め id と URL の紐づけがされている前提でシステムを試作したが，実際は名前解決サーバへの登録の仕組みが必要である．つまり，id を用いてシグナルクラスを生成する際，その id が解決サーバに登録されてなかったら自動的に登録する仕組みを考える必要がある．例えば，使用するデータベースの URL はその id を保持する利用者を決めることが前提とするなら．id が名前解決サーバに登録されていなければ URL の入力を促し，id に紐づけを行うなどの仕組みを作ることが考えられる．また，今回の研究では id を各自で定めていたが，id を自由に決めると衝突が起こる可能性があるため，id 管理の機構が別途必要になると思われる．

### 5.2 非オープンな時系列データ

現状，永続時変値が参照する時系列データは公開が前提となっているが，時系列データには当然秘匿したい情報も含まれていると考えるられる．このような非オープンな時系列データに対してアクセスする場合，シグナルクラスインスタンスに何らかの認証情報を持たせることが必要であると考えられる．

### 5.3 既存の時系列データから永続シグナルへの変換仕様

本研究では，非常にシンプルな時系列データベースを用いており，各列のデータ型もあらかじめ定義している．一方で既存の時系列データは，通常 Excel の表など多様な形式で提供される．これらを本研究の枠組みで利用するには，それらの形式から永続時変値に変換する仕様を決めておく必要がある．例えば，入れ子を含む表は RDB の表に変換できないので，正規化を行う仕組みが必要である．また，Excel で提供される時系列データに変更があった際，その変更が TimescaleDB に反映されない限り永続時変値からは更新が見えない．このように永続時変値と時系列データの自動反映の仕組みも必要である．

### 5.4 更新の競合

永続時変値が，オープンなリアクティブ基盤として複数のプログラムから参照されると，更新の競合の可能性が出てくる．SignalJ が提案する永続時変値のライフサイクルモデルでは，永続時変値の値を更新するインスタンスは世界に一つだけで（もともとローカルな世界を想定している），ほかの（同じ id を参照する）インスタンスは読み取り専用となっている．この前提に立てば更新の競合は起こらない．しかし現状ではこの前提が成り立つことを保証する仕組みは存在しないため，更新者の唯一性を保証する機構を実現する必要がある．また，そもそも本研究が対象とするオープンなリアクティブ基盤において，この前提が妥当かどうかについても検証する必要がある．

### 5.5 シグナルクラス／時系列データベーススキーマの進化

シグナルクラスに新たな永続シグナルを追加するとデータベースのスキーマを変更しなければならなくなるが，基盤がオープンシステムとして作られている以上，古いスキーマを参照するプログラムも共存することになる．新旧スキーマを共存させる技術はデータベース分野で議論されているところだが [5]，本研究でもそれを取り入れていく必要がでてくるかもしれない．シグナルクラスにサブクラスを作る場合も同様である．

## 6. まとめ

本研究では，Java を拡張した言語 SignalJ を分散化したデータベースに紐づいている時変値の参照に対応させ，オープンなリアクティブシステムの基盤を作ることを目的とし，SignalJ 実行時ライブラリに名前解決機能を追加した．サンプルプログラムを用いて分散化した永続時変値にを参照する SignalJ の実装を行うことで SignalJ がオープンなリアクティブシステムの基盤となりうるものであることを確認したと同時に，また数多くの課題も見つかった．今回は一つのサンプルアプリケーションを用いて提案手法の実現可能性を探ったが，今後は様々な事例をもとに，これらの課題の解決に向けた方針を定めていく必要がある．

**謝辞** 本研究の提案内容について深くご議論いただいた東京工業大学の増原英彦氏と株式会社豆蔵の青谷知幸氏に感謝する．本研究は科研費研究課題 21H03418 の支援を受けたものである．

### 参考文献

[1] Cooper, G. H.: Integrating Dataflow Evaluation into a Practical Higher-Order Call-by-Value Language, PhD Thesis, Department of Computer Science, Brown University (2008).
[2] Drechsler, J., Salvaneschi, G., Mogk, R. and Mezini, M.: Distributed REScala: an update algorithm for distributed reactive programming, OOPSLA'14, pp. 361–

ⓒ 2022 Information Processing Society of Japan

376 (2014).
[3] Elliott, C. and Hudak, P.: Functional Reactive Animation, Proceedings of the 2nd ACM SIGPLAN International Conference on Functional Programming (ICFP'97), pp. 263–273 (online), DOI: 10.1145/258949.258973 (1997).
[4] Halbwachs, N., Caspi, P., Paymond, P. and Pilaud, D.: The Synchronous Data Flow Programming Language Lustre, Proceedings of the IEEE, Vol. 79, No. 9, pp. 1305–1320 (online), DOI: 10.1109/5.97300 (1991).
[5] Herrmann, K., Volgt, H., Pederson, T. B. and Lehner, W.: Multi-schema-version data management: data independence in the twenty-first century, The VLDB Journal, Vol. 27, pp. 547–571 (2018).
[6] Kamina, T. and Aotani, T.: Harmonizing Signals and Events with a Lightweight Extension to Java, The Art, Science, and Engineering of Programming, Vol. 2, No. 3 (online), DOI: 10.22152/programming-journal.org/2018/2/5 (2018).
[7] Kamina, T. and Aotani, T.: An Approach for Persistent Time-Varying Values, Onward!'19, pp. 17–31 (2019).
[8] Kamina, T., Aotani, T. and Masuhara, H.: Signal Classes: A Mechanism for Building Synchronous and Persistent Signal Networks, ECOOP 2021, LIPIcs, Vol. 194, pp. 19:1–19:30 (2021).
[9] Meyerovich, L. A., Guha, A., Baskin, J., Cooper, G. H., Greenberg, M., Bromfield, A. and Krishnamurthi, S.: Flapjax: A programming language for Ajax applications, Proceedings of the 24th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Application (OOPSLA'09), pp. 1–20 (online), DOI: 10.1145/1640089.1640091 (2009).
[10] Myter, F., Scholliers, C. and Meuter, W. D.: Distributed Reactive Programming for Reactive Distributed Systems, The Art, Science, and Engineering of Programming, Vol. 3, No. 3, pp. 5:1–5:52 (2019).
[11] Salvaneschi, G., Hintz, G. and Mezini, M.: REScala: Bridging between object-oriented and functional style in reactive applications, Proceedings of the 13th International Conference on Modularity (MODULARITY'14), pp. 25–36 (online), DOI: 10.1145/2577080.2577083 (2014).
[12] Zhuang, Y.: A lightweight push-pull mechanism for implicitly using signals in imperative programming, Journal of Computer Languages, Vol. 54 (2019).

ⓒ 2022 Information Processing Society of Japan