Vol.2015-SLDM-170 No.29
Vol.2015-EMB-36 No.29
2015/3/7

コンテキスト指向フレームワーク ContextCS による掃除機開発

谷川郁太[†1]　渡辺晴美[†1]

実行時に環境に応じて振る舞いの変更するメカニズムを有し，明示的に環境すなわちコンテキストを扱える技術として，コンテキスト指向プログラミング(Context Oriented Programming: COP)がある．COP はコンテキスト適応型の組込みソフトウェアにとって期待される技術である．そのような組込みソフトウェアの多くは，異機種のデバイスを有し，既存資産を有し，プロダクトライン開発を行う．このような特徴を処理可能にするために，我々は COP の概念を備えた Context CS と呼ぶ C#のフレームワークを開発してきた．本フレームワークは以下の特徴を持つ．(1) 実行時に外部からレイヤを追加，(2) サービスプログラムとレイヤ切り替えプログラムの分離，(3) アノテーションによるレイヤ管理．本稿では，ContextCS を評価するために開発する異機種掃除機システムについて紹介する．

A Vacuum Cleaners System for Evaluating a Context-Oriented Framework ContextCS

IKUTA TANIGAWA[†1]　HARUMI WATANABE[†1]

Context-oriented programming (COP) treats context explicitly and provides mechanisms to dynamically adapt behavior in reaction to changes in context at runtime. Such languages are desirable for context-sensitive embedded software. The characteristics are that the software deals with heterogeneous devices and various contexts, reuses legacy programs and forms product lines. To realize these characteristics, we have developed a C# framework called ContextCS that contains the following features: (1) add new layers from outside the system at runtime; (2) separate a service program from a layer changing program; (3) change layers managed by annotation. To evaluate Context CS, the article presents a heterogeneous vacuum cleaner system.

1. はじめに

近年，スマートフォンに代表されるとおり，環境適応型のソフトウェアが盛んに開発されるようになってきた．このようなソフトウェア開発に適したプログラミング言語に，コンテキスト指向プログラミング言語(Context-Oriented Programming: COP)がある．COP は，Robert Hirschfeld [1] らの研究が最初であり，プログラム実行時に，システムを取り巻く外部環境の状況すなわちコンテキストに応じて，実行時にソフトウェアを再構築し，振る舞いを変化させることが可能である[1][2][3][4][5].

環境適応型の組込みソフトウェアは，多様な環境に加えて，異機種のデバイスを処理する必要がある[6]．図 1 は屋内外の両方で稼働する掃除機システムである．屋外では，GPS で自己位置推定をし，屋内ではエンコーダで行っている．加えて，屋内外では，掃除対象のゴミや障害物，床の状況も異なるため，掃除や走行の処理方法も異なってくる．以上の環境の変化は，ソフトウェアの振る舞いに多大な影響を与えるため，ソフトウェア開発は複雑になる．このような振る舞いに対し，COP は，ソフトウェアの再構築を明示的に扱える．自動掃除機のようなロボットでは，環境に応じたデバイスの調整を COP のレイヤ切り替えで実現できるが，環境をリリース前に十分に把握することは難しい．

COP が，環境適応型の組込みソフトウェアの実践的な開発を支援可能にするためには，環境適応の問題に加え，一般的な組込みソフトウェアの性質を踏まえる必要がある．組込みソフトウェアは，既存資産を基盤に開発し，プロダクトラインを伴うのが一般的である．

以上から，筆者らは，環境適応型の組込みソフトウェアの性質として，環境適応に応じた異機種デバイスを扱えること，既存資産を扱えること，さらにプロダクトラインを扱えることを目指し，下記の特徴を備えた ContextCS と呼ぶ C#の COP フレームワークを開発してきた．

(1) 実行時に外部からレイヤを追加：異機種デバイスや多様な環境に対応するシステムは，その状況をリリース前に十分に把握することは困難である．以上の問題を本特徴により解決する．

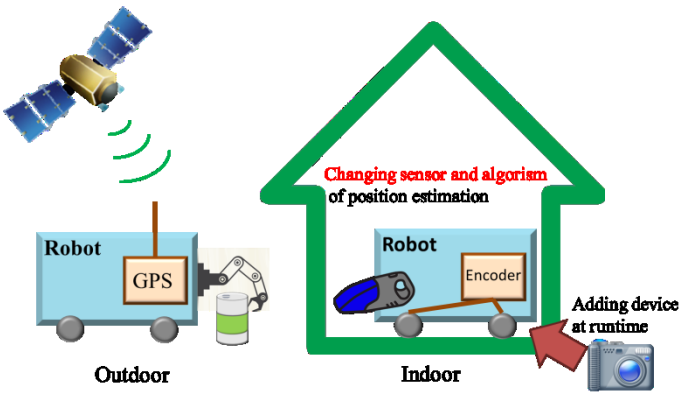

図 1　屋内外で稼働する掃除機ロボット

Figure 1　A vacuum cleaner for working indoor and outdoor

†1 東海大学大学院 情報通信学研究科
Tokai University Graduate School of Information and Telecommunication Engineering

Vol.2015-SLDM-170 No.29
Vol.2015-EMB-36 No.29
2015/3/7

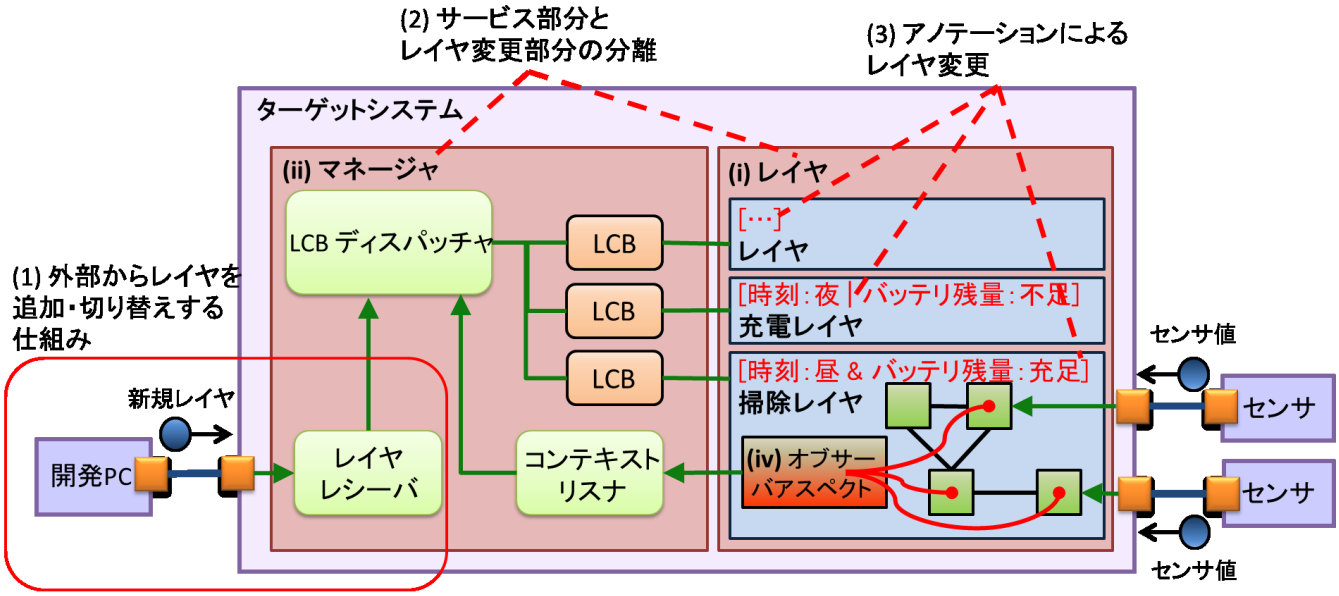

図 2　ContextCS の構造

Figure 2　A structure of ContextCS

(2) サービスプログラムとレイヤ切り替えプログラムの分離：既存資産を流用可能にするために，サービスを提供するプログラムを変更しないようにする必要がある．以上の問題を本特徴により解決する．

(3) アノテーションによるレイヤ管理：プロダクトラインおよび異機種デバイスに対応するために，レイヤにアノテーションを付す．

以上の特徴を評価するために，本稿では，異機種掃除機システムについて紹介する．異機種掃除機システムは，2 台の異なるメーカーの掃除機ロボットを 1 つのシステムで動作させる．本システムでは，異なるデバイスを有す異機種掃除機が，環境の変化に応じ，外部から送付するレイヤに切り替わることを評価することを目的とする．

本研究は，enPiT-EMB/PEARL[7]における自動掃除機ロボットの開発に携わることで着想を得た．また，本研究と関連し，これまで 3 件の発表を行った．まず，外部からプログラムを送り，アノテーションで指定した箇所を実行時に書き換える仕組みを提案した[8]，次に，コンテキスト指向の概念を考慮したフレークワークを提案した[9]．また，その着想の背景について発表した[10]．

以下，2 章では，ContextCS フレームワークについて紹介し，3 章では，異機種掃除機システムについて紹介する．

2. ContextCS フレームワーク

本章では，COP の概念を有す C#フレームワーク ContextCS について述べる．以下，2.1 では構造，2.2 ではコンテキスト適応のためのプログラムについて紹介する．

2.1 フレームワークの構造

本フレームワークは 1 章で挙げた通り，以下の 3 つの特徴を持つ．(1)実行時にシステムの外部から新たなレイヤを追加，(2)レイヤ変更プログラムとサービスプログラムの分離，(3)アノテーションによるレイヤ変更．図 2 に本フレームワークの構造を示す．特徴(1)のために，フレームワークには「レイヤレシーバ」が含まれる．レシーバが外部から新たなレイヤを受け取ると，LCB ディスパッチャがレイヤ切り替えを行う．特徴(2)を提供するために，フレームワークは「レイヤマネージャ」部分と「レイヤ」部分に分かれる．尚，分割するにあたり，各レイヤにオブザーバアスペクトを配置する．

環境の変化はセンサ群から得た値の変化であるが，センサから得た情報を監視しているのは，レイヤに記されているサービスを提供するプログラムである．センサがある特定の値になるとレイヤが切り替わることになるが，それを監視しているのはサービスプログラムということになる．切り替えのタイミングを，サービスプログラムに散在させないために，オブザーバアスペクトを配置することで，サービスプログラムに必要な監視と，レイヤ切り替えに必要な監視を分離する．

特徴(3)を提供するために，レイヤにアノテーションを付す．レイヤ名により，プロダクトラインや異機種デバイスに対応することも可能であるが，直行したプロダクトラインスコープ等を扱うのには不十分である．

2.2 コンテキスト適応のためのプログラム

図 3 はコンテキスト CS フレームワークにおけるユーザプログラムのサンプルである．図 3 における(a)から(c)のプログラムはレイヤプログラムとリスナープログラムのユーザ記述部，LCB ディスパッチャのユーザ記述部である．特徴(3)を実現するために，アノテーションを(a)の(iii)のようにレイヤに付ける．レイヤ切り替えの流れは以下である．初めに，(a)レイヤの(ii)オブザーバアスペクトがマネージャ部分に切り替えの判断を依頼する．次に，(i) リスナープログラムはセンサ値や状態から環境要因を評価する．最後に，

Vol.2015-SLDM-170 No.29
Vol.2015-EMB-36 No.29
2015/3/7

```
[LayerType("時刻:昼 & バッテリ残量:充足")] // (iii) 分類アノテーション
class CleaningLayer : Layer {                  // (i) レイヤ
    private RobotControler _Controler;
    public override void LayerMain() {
        RegisterObserverAspect(new CleaningObserverAspect());
        _Controler = Layer.CurrentLayer.Create<RobotControler>();
        while (true) {
            …
            Controler.Move(…);
        }
    }
    public class RobotControler {
        public void UpdateSensor() { … }
        public void Move(short speed, short radius) { … }
        public void EstimateSelfPosition() { … }
        …
    }
    //(iv) Aspect
    public class CleaningObserverAspect : ObserverAspect {
        [Include(typeof(RobotControler), "UpdateSensor")]
        [Call(Advice.After)]
        public void Weave([JPContext]Context ctx, params object[] args) {
            RobotControler controler = (RobotControler)ctx.Instance;
            Notify(controler);
        }
    }
}
```

(a) レイヤプログラム

```
class CheckingCleaningContext : ContextListener {
    [ListeningMethod()]
    public object CheckContext(RobotControler controler) {
        var result = new ContextSituation();
        if (DateTime.Now.Hour >= 18 || …) result.Time = Time.Night;
        if (controler.Battery <= 30) result.BatteryState = BatteryState.Low;
        if (controler.GetState() == …)  result.StoppingRobot = true;
        return result;
    }
}
public class ContextSituation {
    public Time Time;
    public BatteryState BatteryState;
    public bool StoppingRobot;
}
```

(b) コンテキストリスナのユーザ記述プログラム

```
class RobotSwappingLayer : SwappingLayer {
    public override void EvaluateSwapping(Context context) {
        if (context.ListeningData.Time == Time.Night)
            SwapLayer("時刻:昼", "時刻:夜");
        if (context.ListeningData.BatteryState == BatteryState.Low) …
    }
    public override void EvaluateGuardCondition(Context context) {
        if (context.ListeningData.StoppingRobot == true)
            SwappingLayerFlag = true;
    }
}
```

(c) LCBスケジューラのユーザ記述プログラム

図 3　コンテキスト適応に関するユーザプログラム

Figure 3　User program for context adaptation

(c) LCB ディスパッチャでその結果を用いてレイヤ切り替えを行う．(iv)オブザーバアスペクトは C#のアスペクト指向環境 LOOM.NET[11]で実現する．オブザーバアスペクトを用いる理由は，レイヤ切り替えの評価を行う処理が散在することを防ぐことと，サービスプログラムの開発担当者がレイヤ入れ替えについて意識せずに実装できるようにするためである．

レイヤ切り替え時の処理としては，現在実行中のレイヤを中断し，新たなレイヤを(a)のレイヤプログラムの LayerMain メソッドから実行する．一度切り替えられたレイヤが再度有効になったときは，前回中断したところから実行し直す．これによって，サービスの切り替えを容易にする．

(b)で示したプログラムはレイヤごとのセンサや状態の違いを吸収するためのものである．レイヤによって異なるデバイスを用いる場合，時間や場所といった環境要因を判断する方法が異なる．そのため，(b)ではそれぞれの環境要因を表すための ContextSituation というクラスを用意し，各レイヤのセンサ情報や状態をこのクラスに変換する．これにより，(c)の LCB ディスパッチャでは，レイヤごとのセンサ等の違いを意識することなくレイヤ切り替えを行うことができる．

(c)は(b)の結果からレイヤ切り替えを判断する．切り替えは EvaluateSwapping メソッド中で SwapLayer を呼び出すことで切り替える．このメソッドで指定する値は各レイヤに付けたアノテーションで，レイヤでグループ分けした単位で切り替えが行われる．サンプルでは時刻が昼のレイヤが夜のレイヤに切り替えられる．EvaluateGuardCondition は実際に切り替えて良いか判断するためのメソッドで，このメソッド内で SwappingLayerFlag 変数を真とした時点で，EvaluateSwapping の切り替えが行われる．

3. 掃除機ロボット開発への適用

本章では，ContextCS を評価するために開発する異機種掃除機システムについて紹介する．

組込みではプロダクトやバージョンの違いから，異機種のロボットを動作させる場合がある．また，バッテリの残量によっては充電が必要となり，時間帯によっては騒音の関係から掃除をしていることが望ましくない場合がある．上記のような環境要因に応じてサービス内容を変更するシステムを ContextCS の適用対象として考える．ソフトウェア開発において，前もって全ての環境要因を網羅することは困難であることから，特に本システムは特徴(1) 「実行時にシステムの外部から新たなレイヤを追加」による効果を確かめることを目的とする．

図 5 にシステムの構造を示す．本システムは異機種のロボットが制御 PC と繋がっており，それぞれが制御 PC からの指示に従って動く．加えて，バッテリ残量や時刻に応じて掃除を中断し充電を行う．本適用例では，特徴(1)の評価を行うために，ロボットが台に乗り上げてしまい動けなくなってしまうトラブルに開発者が後から気付いた場合を想定している．図 5 では，開発 PC から開発対象のシステムへ復帰のための新規プログラムを送っている．これにより，

Vol.2015-SLDM-170 No.29
Vol.2015-EMB-36 No.29
2015/3/7

後から気が付いた環境要因に対応することを，システムを停止させること無く行うことが可能となる．

図 5 は適用システムのコンテキスト適応の仕組みを示す．本システムはコンテキストリスナが環境要因として時刻が夜であること，バッテリ不足であることを評価し，LCB ディスパッチャに通知する．LCB ディスパッチャは受け取った環境要因から具体的にどのレイヤに切り替えるのかを決定する仕組みとなっている．開発 PC から新規プログラムを送ることによって，想定する環境要因や切り替えられるレイヤを追加することが可能である．この仕組みにより，コンテキスト指向システムのデバッグを容易にすることを見込める．

筆者らは，これまで enPiT-EMB/PEARL の取り組みにおいて，2 台の同機種の掃除機システムを協調動作するシステムを開発してきた．図 7 は 2 台の同機種の掃除機システムが動作している様子であり，図 8 はマップ作成システムであり，掃除機が稼働した箇所を表示した図である．2013 年度と 2014 年度は，異なるメーカーの掃除機を開発した．これらの経験をもとに異機種 2 台である本システムを開発する．2013 年度，2014 年度の機能はほぼ同等であることから，特徴(2)の既存資産，および(3)のプロダクトラインと合致する．

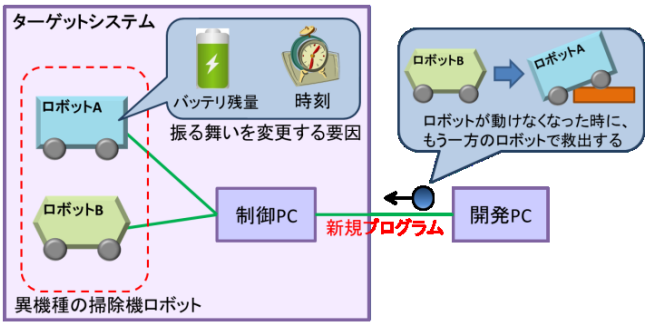

図 4　異機種掃除機ロボットシステム

Figure 4　A heterogeneous vacuum cleaner system

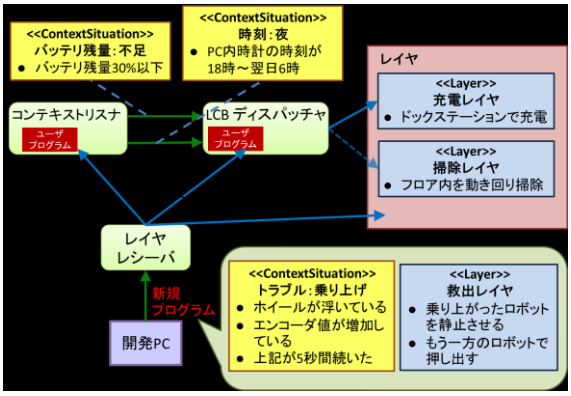

図 5　コンテキスト適応の仕組み

Figure 5　A mechanism of context adaptation

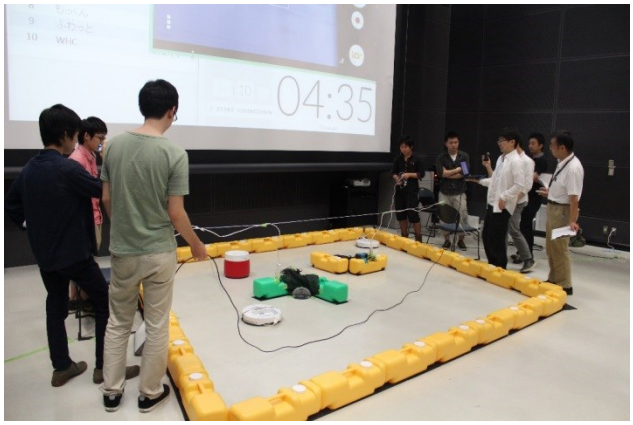

図 6　これまでの掃除機ロボット開発

Figure 6　vacuum cleaner system

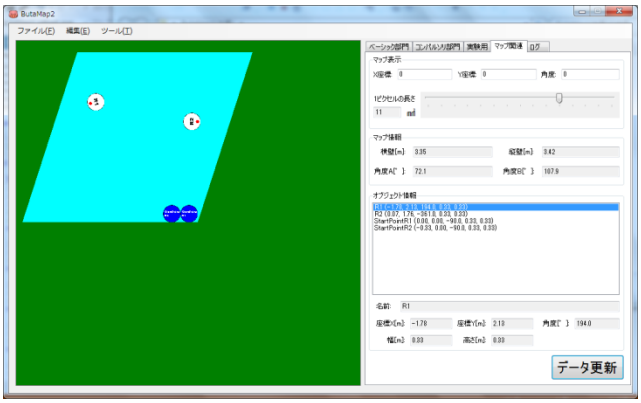

図 7　マップ作成システム

Figure 7　Map viewer

4. おわりに

本稿では，COP の概念を実現するための C#フレームワーク ContextCS および，それを評価するための異機種掃除機システムについて紹介した．ContextCS は，(1) 実行時に外部からレイヤを追加，(2) サービスプログラムとレイヤ切り替えプログラムの分離，(3) アノテーションによるレイヤ管理の特徴を持つ．これらの特徴により，環境適応型の組込みソフトウェアに対し，様々な環境および異機種の問題，既存資産の問題，プロダクトライン開発に貢献することを述べた．

異機種掃除機ロボットでは，2 台の異なるメーカーの掃除機ロボットを 1 つのシステムで動作させ，異なるデバイスを有す異機種掃除機が，環境の変化に応じ，外部から送付するレイヤに切り替わることを評価することを紹介した．

今後は，異機種掃除機ロボットシステムの開発を通し，ContextCS を洗練し，実践的な開発環境の構築に取り組んでいきたい．

Vol.2015-SLDM-170 No.29
Vol.2015-EMB-36 No.29
2015/3/7

参考文献

1) R. Hirschfeld, P. Costanza and O. Nierstrasz: Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp. 125-151, 2008.
2) M. Appeltauer, R. Hirschfeld, J. Lincke,: Declarative Layer Composition with the JCop Programming Language, Journal of Object Technology, Vol. 12, No. 4, 2013.
3) R. Hirschfeld, H. Masuhara and A. Igarashi: L: Context-Oriented Programming With Only Layers, COP'13, 2013.
4) S. Gonzlez, K. Mens, M. Col acioiu and W. Cazzola: Context Trait, AOSD2013, pp. 209-220, 2013.
5) M. Appeltauer, R. Hirschfeld, H. Masuhara, M. Haupt,and K. Kawauchi: Event-based Software Composition in Contextoriented Programming, "In Proceedings of the 9th International Conference on Software Composition, LNCS6144, pp. 50-65", 2010.
6) T. Ruiz-Lpez, C. Rodrguez-Domnguez, M. J. Rodrguez, S. F. Ochoa, J. L. Garrido: Context-Aware Self-adaptations: From Requirements Specification to Code Generation, Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction, LNCS 8276, pp. 46-53, 2013.
7) 分野・地域を越えた実践的情報教育協働ネットワーク組込みシステム分野（enPiT-Emb PEARL） http://emb.enpit.jp/
8) 谷川郁太, 小倉信彦, 菅谷みどり, 渡辺晴美 : 組込みソフトウェアプロトタイプ開発のためのプログラム動的書き換え, 電子情報通信学会技術研究報告. CPSY, コンピュータシステム 113(497), pp.205-209, 2014.
9) 谷川郁太, 小倉信彦, 菅谷みどり, 渡辺晴美 : コンテキスト指向プログラミング実現に向けた実行時プログラム書き換えフレームワークの提案, 組込みシステムシンポジウム 2014 論文集, pp. 84-89, 2014.
10) Ikuta Tanigawa, Harumi Watanabe, Midori Sugaya, Kenji Hisazumi : A Case Study: How to Find and Reify a Research Theme on Project Based Learning for Master's Course Education, SEEW2014 (ASPEC Workshop), 2014.
11) A. Rasche, W. Schult and A. Polze: Self-Adaptive Multithreaded Applications - A Case for Dynamic Aspect Weaving, ARM '05 Proceedings of the 4th workshop on Reective and adaptive middleware systems, Article No.10, 2005.